Universidad Zaragoza
1542

Trabajo Fin de Grado

Captura de múltiples señales analógicas para su posterior procesado utilizando un Cortex M4

Capture of multiple analog signals for further processing using a Cortex M4

Autor

Alberto Bretón Vicente

Director

Isidro Urriza Parroqué

ESCUELA DE INGENIERÍA Y ARQUITECTURA

Captura de múltiples señales analógicas para su posterior procesado utilizando un Cortex M4

Resumen

El presente proyecto consiste en el diseño de un sistema electrónico digital que permita la digitalización de señales analógicas utilizando un microprocesador basado en cortex-M4.

Se ha desarrollado una Capa de Acceso Hardware (HAL) que permite la configuración y control de los periféricos utilizados en este sistema. En el diseño y programación de la HAL, se ha tenido en cuenta la portabilidad. De cara a poder utilizar estas funciones en distintos proyectos que requieran del uso de alguno de los periféricos utilizados.

La HAL es gestionada por una capa de control, que corre sobre el microprocesador. Esta capa, se comunica con una aplicación implementada en el ordenador. Haciendo de la obtención de datos, algo transparente para el usuario.

Además, debido a la importancia del consumo energético en la industria actual. Se han utilizado distintas técnicas para reducir el mismo, haciendo que solo los periféricos imprescindibles consuman energía y que el microcontrolador participe lo mínimo posible en el sistema.

Por último, las herramientas desarrolladas pretenden ser intuitivas y fáciles de utilizar. Se ha elaborado una documentación clara y concisa sobre las implementaciones realizadas. Para ello, se ha hecho uso de un generador de documentación muy extendido en el ámbito industrial llamado Doxygen.

Capture of multiple analog signals for further processing using a Cortex M4

Abstract

This project consists of the design of a digital electronic system that allows the digitization of analog signals using a microprocessor based on cortex-M4.

A Hardware Access Layer (HAL) has been developed that allows the configuration and control of the peripherals used in this system. Portability has been taken into account in the design and programming of the HAL. In order to be able to use these functions in different projects that require the use of any of the peripherals used.

The HAL is managed by a control layer, which runs on top of the microprocessor. This layer communicates with an application implemented in the computer. Making data collection transparent for the user.

Also, due to the importance of energy consumption in today's industry. Different techniques have been used to reduce it, making only the essential peripherals consume power and the microcontroller participates as little as possible in the system.

Finally, the tools developed are intended to be intuitive and easy to use. A clear and concise documentation has been produced on the implementations carried out. To do this, use has been made of a very widespread documentation generator in the industrial field called Doxygen.

Índice

Índice De Figuras

Índice De Tablas

Capítulo 1: Introducción

1.1 Planteamiento y aspectos tecnológicos

El proyecto surge como solución a un problema propuesto específico, la digitalización de señales biológicas en respuesta a ciertos estímulos externos, estudiado.

La respuesta puede capturarse con múltiples electrodos, por lo que se precisa de varios canales de conversión. Además, de espacio suficiente donde almacenar los datos convertidos, de hardware que permita la comunicación con otros dispositivos para su posterior procesado. Y por último, buscando que el microprocesador se ocupe de otras funciones, de un periférico que transporte los datos dentro del dispositivo sin la intervención del mismo.

Aparecen dos opciones, utilizar placas específicas para este tipo de aplicaciones, como las placas desarrolladas por Intan [1]. Placas basadas en FPGA interconectadas a otros periféricos mediante puertos SPI, en particular la placa de Intan estudiada cuenta con unos conversores analógico-digital de 16 bits de resolución. La otra opción, buscar las características deseadas en otras placas de la industria basadas en microprocesadores. Se ha optado por esta segunda opción, empleando el FM4-S6E2CC de Cypress.
El microprocesador seleccionado, cuenta con 3 conversores analógico-digital con 32 canales, 8 canales DMA que permite mover datos entre los distintos periféricos, conexión con una memoria RAM externa y múltiples opciones de comunicación con el exterior. Como son, USB, Ethernet o puerto serie [2].
Como podemos ver, cuenta con soluciones para las necesidades básicas del proyecto y, pese a perder resolución frente a las placas específicas, tiene una clara ventaja en cuanto al coste.
También cuenta con periféricos y funciones que permiten desarrollar técnicas de reducción del consumo.

El procesado de los datos puede ser realizado por el propio microprocesador, o bien transmitir los datos sin procesar y procesarlos en un dispositivo externo. Durante el desarrollo del proyecto se han tenido presentes ambas opciones y se ha programado de tal forma que la CPU quede libre para realizar ella misma el procesado. Aún con esto, al no saber que procesado se desea realizar a los datos obtenidos, se ha realizado la segunda opción, transmitiendo los datos sin procesar a un ordenador. Debe implementarse una aplicación que se comunique con la placa seleccionada para obtener y reconstruir los datos. Se ha escogido Matlab para realizar dicha tarea.
La aplicación devolverá los datos de la manera correcta, proporcionando un vector por canal utilizado. Para indicar las acciones que debe realizar el sistema, el usuario tendrá el control de una interfaz gráfica generada con la herramienta AppDesigner de Matlab.

1.2 Objetivos

1. Implementar un sistema electrónico digital para la digitalización de señales analógicas.

2. Elaborar una HAL (capa de acceso hardware), con funciones simples que permitan utilizar los periféricos del dispositivo.

3. Aportar una documentación de la HAL clara y sencilla.

4. Emplear técnicas que permitan reducir el consumo del dispositivo, para poder así, alimentarlo con una batería de ser preciso.

5. Desarrollar una aplicación de control que gestione el uso de la HAL y realice las distintas tareas.

6. Implementar una aplicación en el ordenador que reconstruya los datos obtenidos y se comunique con la del microprocesador para obtener dichos datos.

7. Dotar al usuario de una interfaz gráfica que permita la interacción con la app del ordenador de la manera intuitiva.

1.3 Alcance

Este proyecto pretende construir un sistema electrónico digital para la digitalización de señales analógicas, haciendo hincapié en la optimización de recursos, la funcionalidad, la portabilidad y la claridad de la documentación aportada.

El sistema debe aprovechar y utilizar de manera eficiente los periféricos de la placa seleccionada, dejando además al microprocesador con la menor carga de cómputo posible. Dando la opción, de utilizarlo en otras funciones.
Diseñar una aplicación sobre Matlab, encargada de reconstruir los datos y mostrarlos de la forma adecuada para su posterior procesado. Para controlar las acciones del sistema, el usuario dispondrá de una interfaz gráfica que indicará a la aplicación las acciones a realizar.

Por último, estudiar y aplicar las distintas técnicas de reducción de consumo, disponibles en la placa seleccionada, contribuyendo al desarrollo sostenible de la tecnología.

1.4 Organización del proyecto

Esta memoria se encuentra organizada en una serie de cinco capítulos en los que se detallan las laboresrealizadas para alcanzar los objetivos planteados en este Trabajo de Fin de Grado.

En el presente capítulo, se detallan los objetivos a alcanzar, el problema propuesto y la estructura de la solución implementada. Se hace un pequeño inciso en la motivación y aspectos tecnológicos del proyecto para hacer comprender mejor la solución finalmente implementada.

En el capítulo 2, se profundizará en los elementos del sistema que forman parte del microcontrolador. Se comentarán las labores que realizan los distintos periféricos empleados y las distintas funciones de la HAL, creada para acceder y controlar los periféricos. Además, se incluye un apartado explicando el método utilizado para reducir el consumo del dispositivo.

En el capítulo 3, se abordarán todos los temas referidos a la aplicación desarrollada en el ordenador. Detallando el protocolo de comunicación entre dicha aplicación y el microcontrolador. Y explicando la composición y características de la interfaz gráfica que permite hacer uso, al usuario, del sistema.

En el capítulo 4, se mostrará la verificación funcional del sistema completo.

En el capítulo 5, se detallan las conclusiones que se han extraído a lo largo del trabajo contrastándolas con los objetivos planteados en el primer capítulo. Y se analizan las posibles vías de desarrollo a futuro para mejorar o incluir funcionalidades al sistema.

Un apartado bibliográfico donde se encuentran las referencias utilizadas para la elaboración de la memoria.

Anexos, donde se aporta la documentación desarrollada. A la que se alude, desde la memoria, en caso de ser preciso conocer más información referente a alguno de los puntos tratados.

Capítulo 2: Microprocesador

En este capítulo se analizan las características y funcionalidades de las capas del sistema programadas en el microprocesador. Antes de abordar la estructura a capas, se analizarán aspectos relativos al dispositivo empleado.

Como he dicho en la introducción, se optó por utilizar el "FM4-S6E2CC Pioneer Kit", basado en un microprocesador de Cortex M-4 de ARM.
ARM ofrece diversas familias de microprocesadores orientadas a multitud de tareas en función de las necesidades de cálculo, consumo energético, periféricos requeridos…Dentro de estas familias, se encuentra la familia Cortex M- de procesadores, implementada para microcontroladores en los que se busca un equilibrio entre potencia computacional, bajo coste y bajo consumo [3]. El Cortex M4 se enmarca en este grupo como el primer procesador de la familia en incluir características DSP (Digital Signal Processor) [4].
Las funciones DSP del microcontrolador, podrían ser necesarias en caso de transmitir los datos ya procesados. En este caso, como no conocemos que se quiere realizar con los datos, los devolveremos en Matlab para poder realizar la acción deseada.

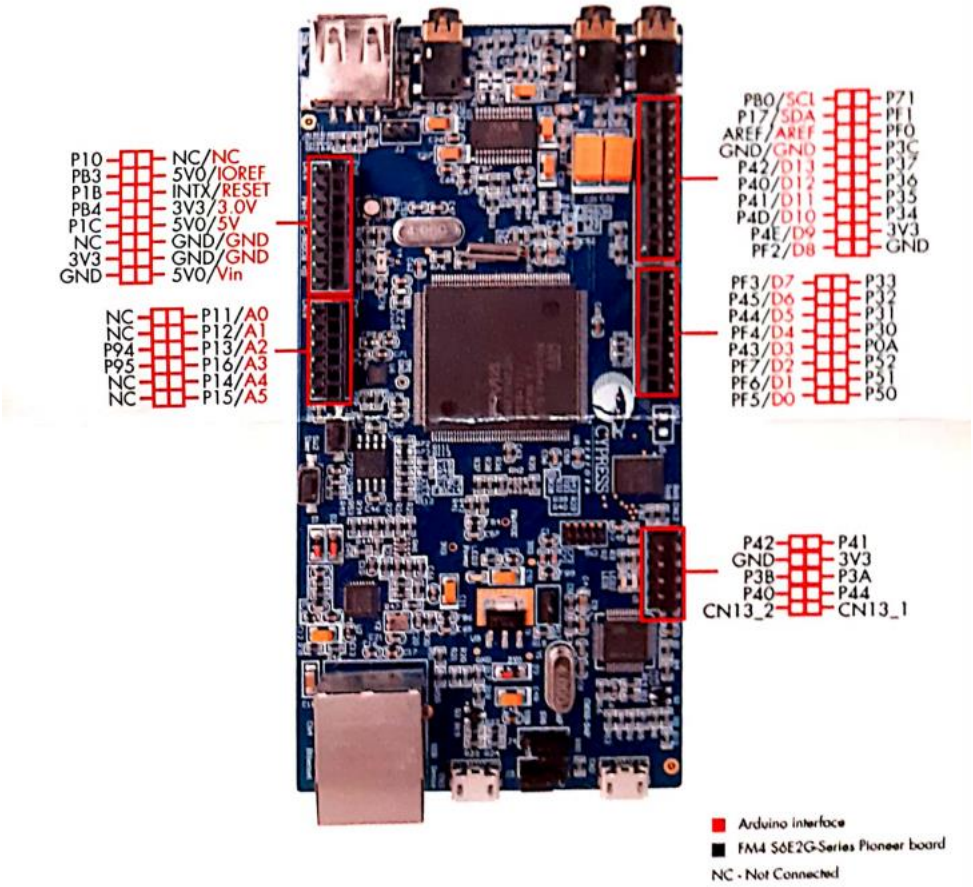

Figura 1. Cypress FM4-176L-S6E2CCAJ0A-ETH.

La placa elegida es la Cypress FM4-176L-S6E2CCAJ0A-ETH, basada en un microprocesador Cortex M-4 que cuenta con una gran cantidad de periféricos.

2.1 Estructura a capas

La programación por capas constituye un modelo en el desarrollo de software en el que se busca que todos los códigos que componen un sistema software, se encuentren desacoplados entre ellos, evitando la dependencia de unos con otros. De esta forma, en caso de requerirse algún cambio en el funcionamiento del software o de aparecer algún error, este solo afectará a la capa correspondiente, evitando tener que revisar el código fuente de otros módulos para corregir el error [5]. Además, facilita el cambio de hardware si se desease cambiar la placa.

El microprocesador gestiona los recursos hardware encargados de la obtención de datos y una vez obtenidos los envía al ordenador. Está programado como una estructura de capas mostrada en la siguiente figura.

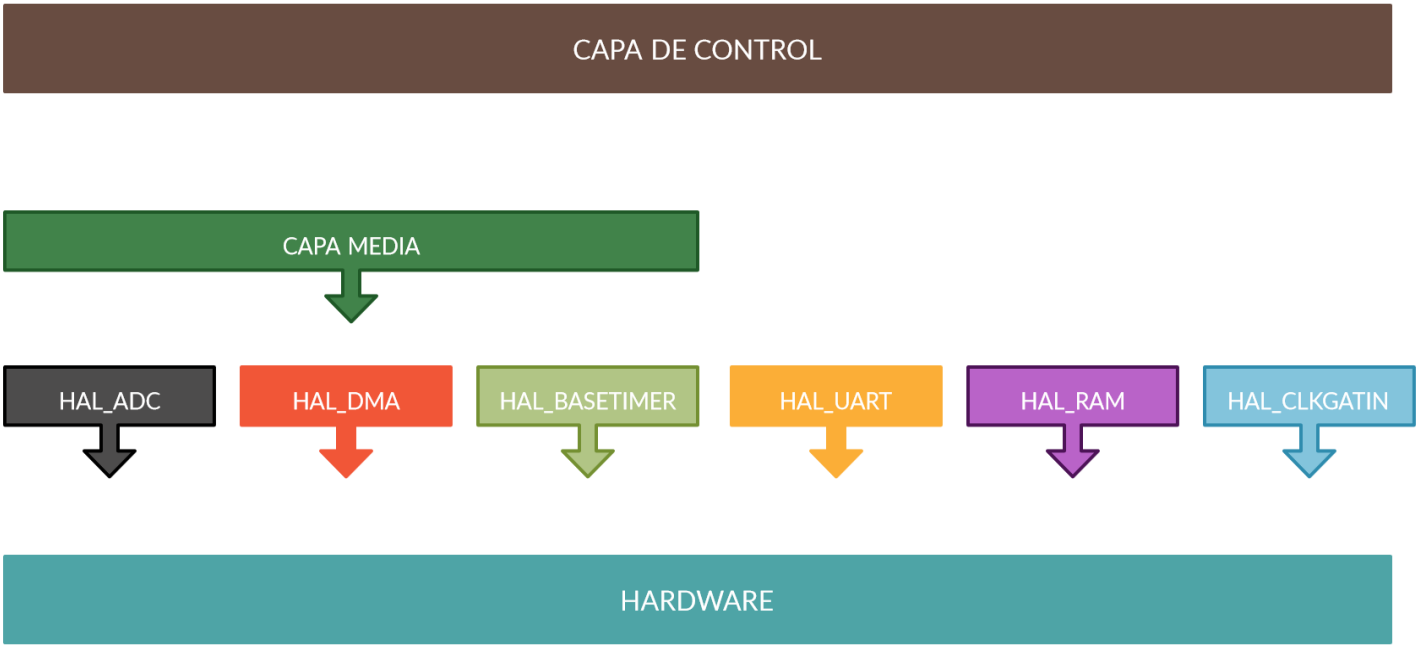

Figura 2. Estructura a capas del microprocesador.

En la parte superior de la estructura encontramos la capa de control, se ocupa de decodificar los mensajes recibidos desde el ordenador y ordenar las acciones pertinentes. Al iniciar la aplicación ordena ejecutar las funciones de inicialización de las HAL (Capa de Acceso al Hardware de sus siglas en inglés “Hardware Abstraction Layer”) de la RAM, la UART y el CLK_GATING, encargado de reducir el consumo.
Las órdenes relativas a la conversión y transmisión interna de los datos, son gestionadas por una capa intermedia que agrupa las funciones conjuntas de las HAL de los ADC, el DMA y el BASE_TIMER.

Una HAL se define como una serie de capas de programación que permiten a los programadores controlar y configurar el Hardware de un dispositivo electrónico. Estas capas de programación que constituyen la HAL, interactúan entre ellas, compartiendo información y recursos para conseguir nuevas funcionalidades [6].
En este caso, se han desarrollado 6 de manera independiente, con el fin de poder utilizarlas en distintos proyectos donde se usen alguno de los periféricos aquí

utilizados.
Empezando de izquierda a derecha en la Figura 2, tenemos la HAL del ADC. Cuenta con las funciones necesarias para utilizar los conversores analógico-digital, encargados de obtener los datos procedentes de las señales analógicas de las distintas entradas utilizadas.
A continuación, muy relacionados con el ADC, se encuentra la HAL del Base Timer y la del DMA. El Base timer, establece la frecuencia de muestreo a la que van a convertir los ADCs.
El DMA, se encarga de la transmisión de los datos dentro del circuito, desde los conversores a la memoria RAM, donde se almacenan los datos convertidos, y de la memoria RAM al puerto serie de salida.
La HAL de la UART, cuenta con las funciones para crear el puerto serie bidireccional entre el microprocesador y el ordenador.
Por último, el CLK_GATING se ocupa de deshabilitar la señal de reloj de los periféricos no utilizados para reducir el consumo.

2.2 Periféricos

Los periféricos son todos aquellos circuitos digitales que permiten al microcontrolador interactuar con el mundo "exterior". Van desde módulos para habilitar/deshabilitar salidas digitales hasta circuitos que permiten al microcontrolador establecer canales de comunicación con otros dispositivos [7].

Para facilitar la programación de la HAL que hace uso de los periféricos, se ha usado el estándar CMSIS (Common Microcontroller Software Interface Standard). CMSIS, define una serie de Interfaces para la Programación de Aplicaciones en los procesadores de ARM Cortex M- y sus periféricos [8].

CMSIS surgió como una vía para establecer librerías de drivers en los controladores Cortex M, dando lugar a CMSIS core, desde entonces se han desarrollado bloques adicionales especializados en distintos ámbitos.
En general, podemos dividir CMSIS en varias capas como se muestra en la Figura 3.

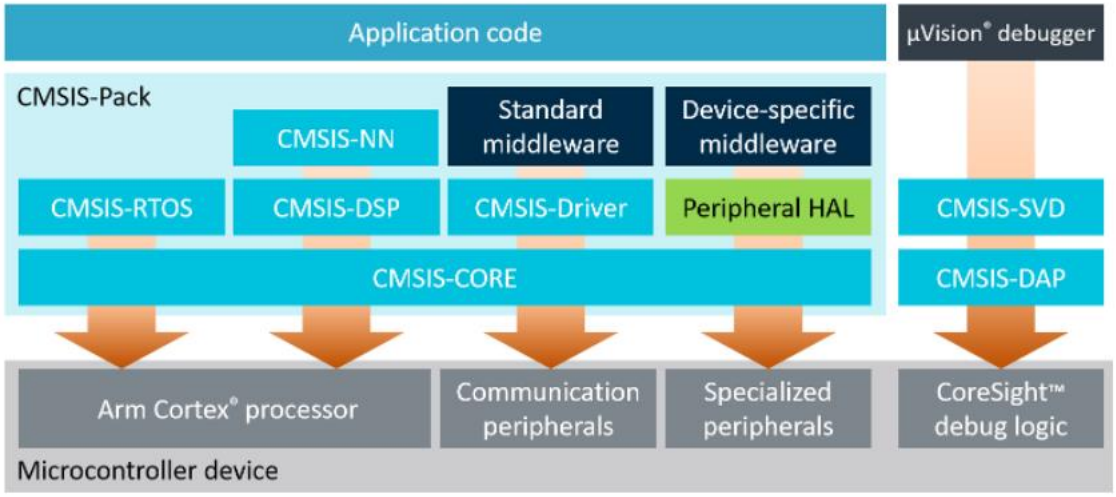

Figura 3. Estructura de capas CMSIS.

Cada bloque ofrece distintas funcionalidades, desde sistemas operativos RTOS (Real

Time Operation System), hasta librerías orientadas a redes neuronales. En este trabajo destaca, empero, CMSIS-Core, que permite acceder a las características de la CPU [9].

A continuación, veamos más en detalle aspectos técnicos de los periféricos empleados y las funciones implementadas en las distintas HAL. Para un análisis más detallado del código y las funciones se remite al lector al apartado de anexos.

2.1.1 ADC

Los conversores analógico digital (ADC del inglés Analog-to-Digital Converters) transforman las señales analógicas en valores digitales para utilizarlos posteriormente en sistemas de procesado o control [10].

Cuento con 3 conversores analógico-digital de 12 bits de resolución y una frecuencia máxima de muestreo de 1MHz para cada ADC. La frecuencia de muestreo se divide entre los distintos canales, del ADC en uso. Debe tenerse presente a la hora de seleccionar el número de canales y la frecuencia de muestro [11]. Por ejemplo, un ADC no puede utilizar dos canales a la frecuencia máxima, divide esta en 500KHz para cada canal.
Para establecer la frecuencia de muestreo, se realiza una conversión en el momento que se detecta un flanco de subida en la señal de reloj generada en el BaseTimer.
Con el fin de maximizar la frecuencia de muestreo en el máximo número de canales disponibles, se va aumentando equitativamente el número de canales utilizados por cada ADC. De tal forma, usando 3 canales se puede convertir en cada uno con la frecuencia máxima, ya que se habilitaría un canal de cada ADC. Así irían aumentando uno a uno los canales hasta llegar a 5 para los ADC 1 y 2 y 6 canales para el ADC0. La siguiente tabla muestra las funciones implementadas para el control de los ADC.

Funciones

uint16_t	**adc_config** (uint16_t n_canales, uint32_t frecuencia)
	adc_config Configura los ADC. Tener en cuenta que ambos parámetros están relacionados ya que los ADC reparten la frecuencia entre los canales activos. Por lo tanto, no se puede tener en un ADC dos canales activos a frecuencia máxima. El número de canales va aumentando equitativamente entre los distintos ADCs, así, podemos usar 3 canales a 1MHz, frecuencia máxima. .
void	**adc_start** (void)
	adc_start inicia la conversión, debe colocarse en último lugar.
Void	**adc_clear** (void)
	adc_clear limpia las memorias de los ADC.

Tabla 1.Funciones HAL ADC.

Se han configurado los ADCs para poder servir a 16 entradas analógicas, ya que todos

los pines mostrados en la Figura1, tienen distintas funcionalidades en la placa de desarrollo utilizada, que se deben usarse de manera forzosa, evitando emplearlos como entradas analógicas.
En la Tabla1 se muestran los pines con posibilidad de usarse como entradas analógicas. Las dos primeras columnas de la tabla corresponden con el localizador y el nombre de cada pin. Las 6 columnas siguientes muestran las posibles funciones que pueden realizar. Por último, la columna de la derecha indica la ubicación del pin dentro de la placa.

LQP176	Pin Name	Alt. Pin Func. 0	Alt. Pin Func. 1	Alt. Pin Func. 2	Alt. Pin Func. 3	Alt. Pin Func. 4	Alt. Pin Func. 5	Alt. Pin Func. 6
94	P10	AN00	SIN10_0	TIOA0_2	AIN0_2	INT08_0		as CN10-3 A3
95	P11	AN01	SOT10_0 (SDA10_0)	TIOB0_2	BIN0_2			as CN10-2 A4
96	P12	AN02	SCK10_0 (SCL10_0)	TIOA1_2	ZIN0_2			as CN10-1 A5
97	P13	AN03	SIN6_1	RX1_1	INT25_1			as CN10-4 A2
98	P14	AN04	SOT6_1 (SDA6_1)	TX1_1				as CN10-5 A1
99	P15	AN05	SIN11_0	TIOB1_2	AIN1_2	INT09_0		fp CN16-7
100	P16	AN06	SOT11_0 (SDA11_0)	TIOA2_2	BIN1_2			as CN7-9
101	P17	AN07	SCK11_0 (SCL11_0)	TIOB2_2	ZIN1_2			as CN7-10
107	P19	AN09	SOT2_0 (SDA2_0)	TIOB3_2	INT24_1	TRACECLK		
103	PB1	AN17	SCS60_1	TIOB9_1	INT08_1			sch PhotoTrans
109	P1B	AN11	SIN12_0	TIOB4_2	INT11_0	TRACED1		as CN9-1
110	P1C	AN12	SOT12_0 (SDA12_0)	TIOA5_2	TRACED2			as CN9-2
111	P1D	AN13	SCK12_0 (SCL12_0)	TIOB5_2	TRACED3			fp CN16-5
116	P1E	AN14	TIOA8_1	INT26_1	MAD10_0			
117	P1F	AN15	RTS5_0	TIOB8_1	INT27_1	MAD11_0		
106	P18	AN08	SIN2_0	TIOA3_2	INT10_0			sch RGB (LED B)
126	P22	AN31	SOT0_0 (SDA0_0)	INT26_0				
125	P23	UHCONX1	AN30	SCK0_0 (SCL0_0)	TIOB13_1			
124	P24	AN29	TIOA13_1	MAD18_0				
123	P25	AN28	RX1_0	INT25_0	MAD17_0			
121	P27	AN27	SIN5_0	INT24_0	MAD15_0			
120	P28	AN26	SOT5_0 (SDA5_0)	MAD14_0				
119	P29	AN25	SCK5_0 (SCL5_0)	MAD13_0				
118	P2A	AN24	CTS5_0	MAD12_0				
104	PB2	AN18	SCS61_1	TIOA10_1	INT09_1			sch RGB (LED G)
102	PB0	AN16	SCK6_1 (SCL6_1)	TIOA9_1				as CN10-6 A0
108	P1A	AN10	SCK2_0 (SCL2_0)	TIOA4_2	TRACED0			sch RGB (LED R)
105	PB3	AN19	SCS62_1	TIOB10_1				fp CN16-6
112	PB4	AN20	SIN8_1	TIOA11_1	INT10_1	TRACED4		
113	PB5	AN21	SOT8_1 (SDA8_1)	TIOB11_1	INT11_1	TRACED5		
114	PB6	AN22	SCK8_1 (SCL8_1)	TIOA12_1	TRACED6			
115	PB7	AN23	TIOB12_1	TRACED7				fp CN16-4

Tabla 2.Pines analógicos de la placa.

Las filas sombreadas corresponden con los pines utilizados como entradas analógicas.

Los distintos colores se deben al mapeo de los mismos dentro de los 3 ADCs. Las sombreadas en verde, corresponden con las posibles entradas analógicas del ADC0. Las naranjas, son los posibles canales a servir por el ADC1. Y por último las amarillas servidas por el ADC2.
Apreciar, que se han configurado el AN08, correspondiente con el led RGB y el AN17, correspondiente con fototransistor. Sobre estos pines no se tiene acceso directo, por lo tanto, para esta placa, en particular, se disponen de 14 canales por los que introducir señales analógicas.

El número de conversiones realizadas es fijo y se reparte entre los distintos canales. Como podemos ver en la columna Alt.Pin.Func.0 de la Tala1, todos los canales analógicos tienen asociado un número del 0 al 31. Siempre se convierte siguiendo el orden de menor a mayor.
Conocer esto es de suma importancia para la posterior reconstrucción, ya que los datos de las distintas señales se almacenarán intercalados.

2.1.2 Base Timer

Este periférico, establece la frecuencia de muestreo de los ADC.

Se le proporcionan los ciclos que debe esperar, de un reloj de 100MHz para generar la señal de inicio de conversión **[12]**.

El cálculo de los ciclos se realiza en la función "ADC_config" antes vista. Depende de la frecuencia de muestreo solicitada por el usuario.

Funciones

void	**ini_BT** (uint16_t cicles)
	ini_BT inicia y configura el Base Timer.
void	**stop_BT** (void)
	stop_BT detiene el Base Timer

Tabla 3.Funciones HAL Base Timer.

Para comprobar el correcto funcionamiento de este periférico, se estableció una frecuencia de muestreo y se hizo variar la salida de un pin digital cada vez que saltase la interrupción del ADC. Introduciendo dicha salida en el osciloscopio, verifiqué que los cambios se producían en las frecuencias deseadas.

2.1.3 DMA

El acceso directo a memoria (DMA, del inglés Direct Memory Access) permite a los periféricos acceder a la memoria del sistema para leer o escribir independientemente de la unidad central de procesamiento (CPU) [13].

En la HAL se configuran 4 canales de DMA. 3 de ellos, del canal 0 al canal 2, se ocupan de la transmisión de los datos, desde los ADC o las entradas externas, a la memoria RAM externa. El canal 3 se ocupa de la transmisión desde la memoria externa a la salida de la UART.

Lo primero que se debe hacer, es seleccionar el tamaño de los datos a transmitir y el tipo de transmisión. En los 3 primeros canales el tipo escogido es el demand mode [14]. Con esta configuración, el DMA se conecta a la interrupción de los ADC, una vez se genera la interrupción de conversión realizada con éxito, el DMA mueve el dato a la dirección correspondiente, en este caso, la memoria externa.

En el canal tres, la opción escogida es la block transfer mode. En este modo, se dividen los datos en un número de paquetes determinados, de un tamaño también determinado, cada vez que el DMA recibe la interrupción de salida del MFS (Multi-Function Serial) envía un paquete. Así se logra, transmitir los datos completamente y sin pérdidas debido a la saturación de la FIFO del MFS.

Posteriormente, se deben establecer las direcciones de origen y destino, indicar si deben ser fijas o móviles. De ser móviles, la dirección aumenta en función del tamaño de los datos transmitidos. En el caso de los ADC, se fijan las direcciones de los mismos y se establece como dirección móvil la dirección de la memoria para llenar el espacio deseado. En el caso de la transmisión a la UART, se fija la dirección destino y se deja variable la dirección origen para ir transmitiendo todos los datos a la UART.

Para dar la opción de introducir los datos usando ADCs externos, de mayor resolución como los empleados en los dispositivos de Intan, se da la opción de cambiar la dirección de origen del canal DMA0 en función del tipo de conversión introducido por el usuario, dando como dirección alternativa al ADC0 el puerto SPI.

Además, hay que seleccionar el número de datos a transmitir, Este valor es fijo en el caso de las transmisiones de los tres primeros canales del DMA y se colocan en serie a lo largo de la memoria. Ocupando los primeros 32Kbytes los datos del DMA0, los siguientes 32Kbytes los datos del DMA1 y los siguientes 32Kbytes los datos del DMA2.

Los datos que transmite el canal 3 del DMA, son por lo tanto, variables en función del número de ADCs que haya convirtiendo.

Funciones

void	**DMA_start** (void)
	DMA_start inicia los canales de DMA que quedan a la espera de las peticiones hardware para iniciar la transmisión.
Void	**DMA_clear** (void)
	Limpia la cola de interrupción del DMA.
Void	**DMA_config** (uint16_t TC, uint32_t DR)
	DMA_config configura todos los canales de DMA a utilizar. Los canales de 0 a 2, este último incluido, se usan para pasar datos desde los ADC a la memoria externa. El canal 3 se utiliza para transmitir los datos de la memoria al UART.
Void	**DMA_stop** (void)
	DMA_stop detiene los canales de DMA.

Tabla 4.Funciones HAL DMA.

En el caso del canal 3, la transmisión no debe comenzar hasta que se haya completado la transmisión de datos a la RAM externa. Por ello, se inicia al recibir la interrupción de los otros 3 canales de DMA y no en la función DMA_start.

Para indicar a la CPU que se ha completado con éxito la transmisión de los datos desde los ADC a la memoria, es necesario habilitar la interrupción CI del DMA. Con el fin de no bloquear el sistema haciendo que la CPU atienda constantemente esta interrupción, hay que resetear manualmente los bits que indican el estado del DMA, limpiando la interrupción [14].

2.1.4 RAM externa

La memoria RAM (Random Access Memory) es utilizada por la CPU y los distintos periféricos para realizar acciones de lectura y escritura. Su nombre proviene del hecho de poder almacenar y leer datos sin necesidad de llevar un orden secuencial. Por lo general es un tipo de memoria temporal, al apagar o reiniciar el sistema, vuelve a estar en blanco [15].

La placa utilizada cuenta tanto con RAM internas al microcontrolador como con una RAM externa. Se decidió usar debido a que su gran capacidad, 256MB, permite

almacenar todos los datos para poder transmitirlos a la UART de manera más sencilla usando el canal 3 del DMA [14].

Utilizarla requiere del uso de ciertos pines, siendo el uso de este periférico, uno de los limitantes del número de entradas analógicas disponibles en la placa [14].

Funciones

void	**FM4_ram_init** (void)
	FM4_ram_init inicializa la RAM externa para ser usada por el DMA.

Tabla 5.Función HAL RAM externa.

Se almacenan, como ya se ha dicho, 32Kbyte de cada ADC. Corresponde con 16000 conversiones. En la digitalización de señales biomédicas del corazón, se suelen utilizar frecuencias de muestreo bastante bajas [16]. Por tanto, suponiendo una frecuencia de muestreo de 1KHz y un canal por cada ADC. El número de datos convertidos corresponde con 16 segundos de cada señal, tiempo que he considerado suficiente para medir la respuesta deseada.
Por lo tanto, el tiempo máximo de señal convertida será de 16 segundos y el mínimo de 3,2 segundos, para las señales de los ADC1 y ADC2, y de 3 segundos para las señales convertidas del ADC0.
De querer aumentarse, sería posible ya que solo se están utilizando 96KB de los 256MB disponibles. Únicamente, habría que cambiar el número de datos a transmitir, seleccionado en cada DMA, realizando los cambios pertinentes en las direcciones de destino para no sobrescribir unos datos con otros.

2.1.5 UART

UART, acrónimo de Universal Asynchronous Receiver-Transmitter. Es un periférico cuya función es convertir los datos recibidos procedentes del bus del PC en formato paralelo, a un formato serie. También realiza el proceso contrario convirtiendo los datos de serie procedentes del microcontrolador de serie a paralelo **[17]**.

Admite comunicación bidireccional y un modo maestro esclavo. Para esta aplicación, se usará la comunicación bidireccional. Configurado el canal, es necesario habilitar, usando NVIC, la interrupción de recepción del MFS (Multi-funtion serial) **[18]**. Con esto la CPU entrará en funcionamiento al recibir un mensaje desde el ordenador y llamará a la función de demodulación de la app del microcontrolador.

La interrupción de transmisión se conecta al canal 3 del DMA. Esta petición se hace cuando el registro de salida del MFS está vacío. Una vez arranca el sistema, el MFS en transmisión lanza su interrupción pidiendo un nuevo dato para transmitir. Como no recibe ningún dato, ya que solo se traspasan una vez son solicitados por el usuario desde la interfaz gráfica, el registro se bloquea y no vuelve a realizar una petición.

La solución a este problema consiste en enviar un dato al registro habiendo habilitado el canal 3 de DMA, logrando que este se desbloquee y solicite los datos al canal 3 del

DMA, que se encuentra a la espera, listo para transmitir los paquetes correspondientes.

Cada dato de 16 bits es dividido en palabras de 8 bits para ser transmitido, ya que el registro de salida del MFS0 es de 8 bits. Las transmisiones son en Little-Endian, enviando primero los 8 bits menos significativo y posteriormente, los 8 más significativos. Debe tenerse presente a la hora de reconstruir los datos.

Funciones

void	**uart_config** (void)
	uart_config configura el transmisor y receptor del UART. Se utiliza la FIFO con capacidad para 64 bytes aunque las transmisiones del DMA se harán en paquetes de 16 bytes= 8 datos (máximo permitido).
Void	**canal_config** (void)
	canal_config configura los pines que vamos a utilizar para crear la conexión puerto serie.

Tabla 6.Función HAL UART.

2.3 Capa media

La capa media comprende una serie de funciones que aglutinan las demás de las HAL más utilizadas. El objetivo es hacer más legible el código realizando el menor número posible de llamadas a funciones de las HAL.

Son paquetes de funciones destinados a tres distintas actividades.

Funciones

uint16_t	**configuracion_hal** (**tipo** type, uint16_t n_canales, uint32_t frecuencia)
	configuracion_hal configura todos los periféricos a utilizar en el proyecto
void	**start** (void)
	start inicio de los periféricos del proyecto
void	**stop** (void)
	stop detención de los periféricos del proyecto

Tabla 7.Función Capa intermedia.

En la función de configuración, se asignan los valores de las variables de las distintas funciones en función de lo solicitado por el usuario.
Se establece una dirección de origen del canal 0 del DMA, dependiendo del tipo de conversión seleccionada, con conversores analógico-digital internos o externos. En función del número de canales solicitados, se establece el número de datos a transmitir por el canal 3 del DMA y se devuelven los ciclos de reloj que debe esperar el Base Timer, en función de la frecuencia de muestreo introducida.

2.4 Capa de control

La aplicación implementada en el microprocesador, gestiona todo el proceso de obtención de datos.
Inicialmente, se encarga de crear el canal de conexión con el mundo exterior, de la inicialización de la RAM externa y de la ejecución la función del CLK_GATIN.

Después, únicamente atenderá a dos interrupciones, la de recepción de mensajes por el canal 0 del MFS (MFS0) y la interrupción de transmisión de dato completada de los canales de DMA que almacenan datos en la memoria RAM.

Cuando se recibe un mensaje nuevo, la CPU atiende a la interrupción de recepción del MFS0. Almacena el valor recibido en la SRAM2, memoria interna del microcontrolador y aumenta el valor de un contador, el valor de este contador se aumenta ante la recepción de cada mensaje y se reinicia al completar la función de decodificación. Al recibir el segundo mensaje, correspondiente con el tamaño del paquete de mensajes transmitido desde el ordenador. Se comprueba si el contador tiene un valor idéntico al tamaño del paquete en transmisión.
Si el valor del contador es igual que el tamaño del paquete transmitido, se llama a la función “decodificación” para realizar las acciones correspondientes a cada comando. Si el valor del contador es menor, se sigue esperando a recibir datos hasta que el contador alcanza el valor del tamaño marcado.

Una vez se llama a la función de “decodificación”. Se transmite el ACK, confirmando la recepción del mensaje. Para formar el ACK, se vuelcan los valores almacenados en la SRAM2 sobre el registro de transmisión del MFS0.
Posteriormente, se comprueba el comando recibido y se realizan las funciones que correspondan.
Si se recibe el comando de apertura o cierre de comunicación, simplemente devuelve el ACK para confirmar a la aplicación del ordenador, que se ha iniciado la conexión o que puede cerrar la conexión con el canal.
Cuando recibe los mensajes de Start o de Stop, el microprocesador lanza las funciones de la capa media, en la que se habilita o deshabilita los distintos periféricos, iniciando o deteniendo el proceso de conversión y transmisión. El comando en el que más trabajo se tiene que realizar es el de configuración, donde se asignan los valores recibidos en el mensaje a las variables de la función de la capa media asociada a la tarea de configuración.

La otra interrupción a la que el microcontrolador debe servir, es la generada por los canales 0, 1 y 2 del DMA cuando se completa la transmisión de datos de los conversores a la memoria. En este momento, se deshabilitan los bit de enable de los

ADC y se resetean los estados de los canales de DMA 0,1 y 2, con el propósito de no alterar los valores de la memoria hasta que se haya completado la transmisión al ordenador.
Además de deshabilitar los bits necesarios, se habilita el bit de enable del canal 3 de DMA, iniciando el volcado de datos de la memoria al registro de transmisión del MFS0. Como se ha dicho, se envía antes un uno para desbloquear la interrupción de transmisión del MFS0.

Cuando se recibe el comando correspondiente con el cierre de la conexión, ejecuta un RESET para devolver a los valores por defecto a todos los periféricos empleados.

2.6 Camino de datos

Se han analizado las distintas capas que intervienen en el desarrollo de la aplicación y las funciones que realizan los distintos periféricos.
Para comprender mejor el funcionamiento conjunto de todos los periféricos y la capa de control. Analizaremos como llevan a cabo las dos acciones del sistema realizadas en la placa, digitalizar los datos y transmitirlos a través del puerto serie.

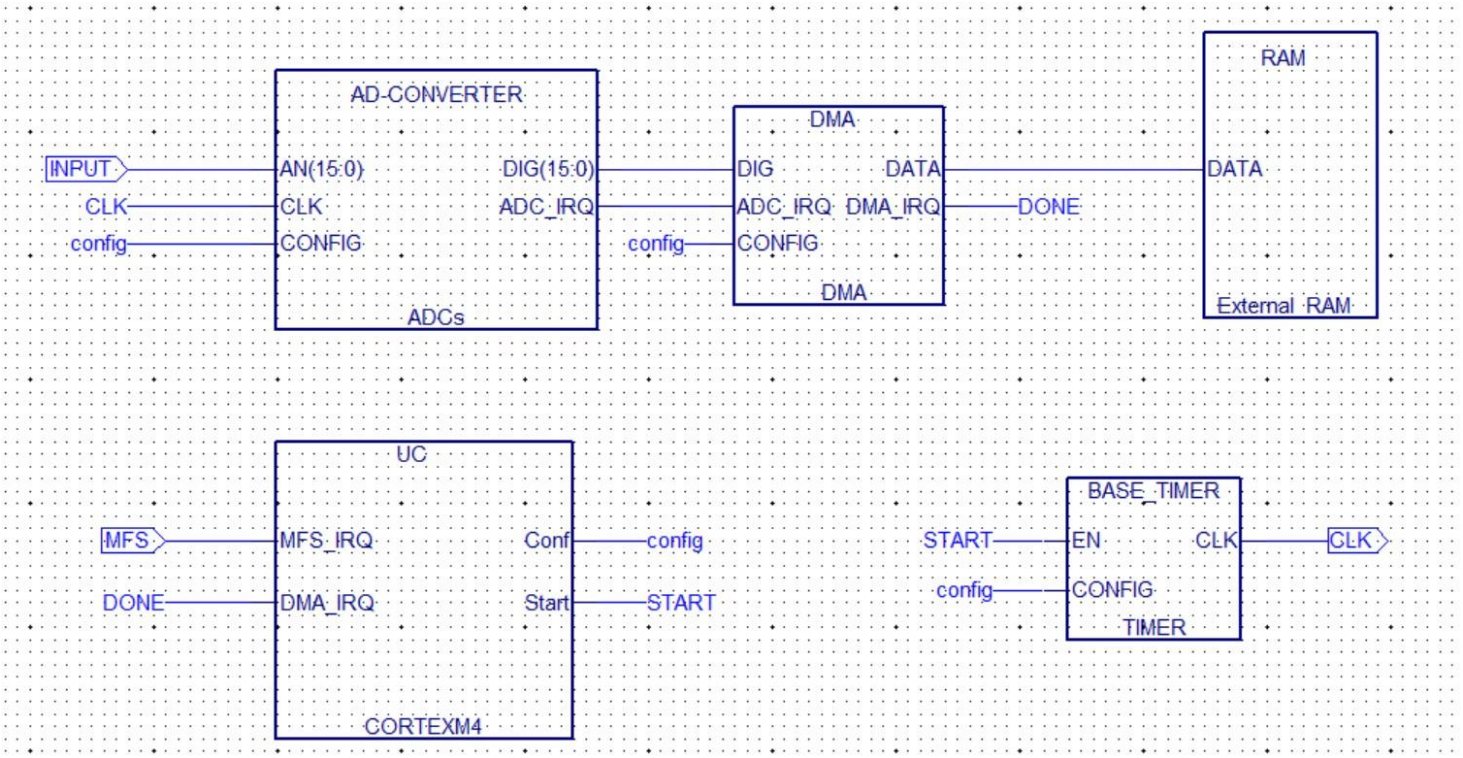

Figura4. Diagrama de bloques digitalización de datos [19].

Comencemos con la acción de digitalizar los datos. En la Figura4 podemos ver los periféricos que intervienen con sus señales determinantes.

Antes de comenzar la acción, el microprocesador debe recibir del puerto serie, la señal MFS, la orden de configurar, generando la señal config. Esta señal, configura todos los periféricos según se haya indicado en el mensaje recibido en el puerto serie.
Una vez configurado, el microcontrolador espera la recepción a través de la misma señal

de la orden de inicio. Ambas señales, son atendidas en la interrupción de recepción del puerto serie. Cuando recibe la orden de inicio, indica al BaseTimer que debe comenzar a contar ciclos de reloj para generar la señal CLK de la frecuencia deseada.

Los conversores analógico-digital realizan una conversión con el flanco de subida de la señal CLK, convirtiendo valores de la señal o señales de entrada INPUT.
Cada vez que convierten un dato, generan una interrupción, la señal ADC_IRQ. El DMA atiende esta señal cogiendo el dato convertido y almacenándolo en la memoria RAM.
El número de conversiones que realizan los ADCs es fijo y una vez se alcanza, los canales de DMA generan su interrupción, indicando a la CPU que debe comenzar la siguiente acción.

Como vemos, el microprocesador solo entra en funcionamiento para atender las interrupciones del DMA y del puerto serie. Podría no ser así, y atender la interrupción del ADC para transmitir los datos directamente o realizar la función de los canales de DMA y almacenar los datos en la memoria RAM. Esto supondría la saturación de la CPU que quedaría atendiendo únicamente las conversiones de los ADCs.
La forma de implementación escogida, deja desahogada la CPU permitiendo, de ser requerido, aprovechar las funciones DSP antes mencionadas de las que dispone el Cortex M4.

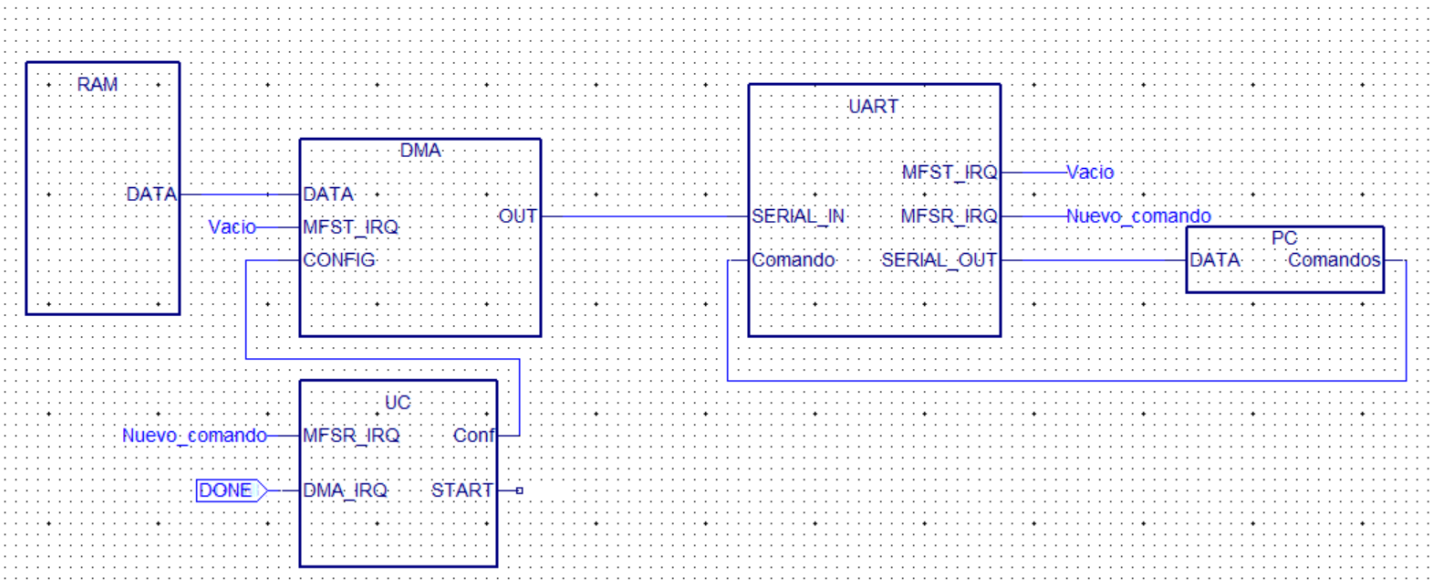

Figura5. Diagrama de bloques transmisión puerto serie [19].

Una vez se ha recibido la señal DONE, se comienza con el proceso de trasmisión de datos. Recuérdese, que al no haber atendido la interrupción de transmisión del puerto serie, MFST_IRQ, en el momento de inicio del sistema, esta queda bloqueada y debe desbloquearse. Para ello, se coloca un 1 en el registro de transmisión, como ya se ha comentado. Tras enviar este 1, al encontrar el registro de transmisión vacío, se genera nuevamente la interrupción MFST_IRQ que ahora atenderá el canal 3 del DMA, que irá volcando los datos de la memoria RAM en el registro de transmisión.

Destacar, ya que en la anterior imagen no estaba presente, la recepción de la señal comandos en el UART, señal que indica la recepción de un paquete con las acciones que debe realizar la CPU. La CPU atenderá la interrupción, MFSR_IRQ e indicará que se debe realizar. Completando la acción de la capa de control.

2.7 Ahorro energético

El circuito integrado empleado está desarrollado, como la mayor parte de la industria mundial, de tecnología CMOS (complementary metal-oxide-semiconductor). La tecnología CMOS utiliza conjuntamente transistores de tipo pMOS y nMOS configurados de tal forma que, en estado de reposo, el consumo energético sea únicamente debido a las corrientes parásitas [20]. Frente a este consumo, como diseñador, no hay nada que pueda hacer.
Por otro lado, el consumo producido por corrientes dinámicas, que aparecen en la conmutación de los transistores, es sobre el que puedo ejecutar las distintas técnicas estudiadas.
La intuición lleva a considerar reducir la frecuencia de reloj, de esta forma se reducen el número de conmutaciones haciendo que el circuito integrado reduzca su consumo. Pero, en la aplicación podríamos requerir de la mayor velocidad posible. Por lo tanto, se prefirió hacer que el microprocesador solo interviniera en los momentos necesarios y retirar la señal de reloj de los periféricos no empleados.

Por un lado, se mantendrá el controlador en Sleepmode hasta que llegue alguna interrupción que le obligue a intervenir. Esto se hace empleando el comando "_WFI()" (Wait for Interrupt) [21], las interrupciones que obligan al microprocesador a tomar parte en el proceso son la de recepción del MFS y las de transmisión completada de los tres primeros canales del DMA.
La interrupción de recepción del MFS se genera al recibir un mensaje nuevo desde el ordenador. Momento en el que la CPU se despierta, almacena el mensaje en la memoria SRAM2 interna y llama la función de demodulación. Lee el mensaje recibido y realiza la función que corresponda. Además, escribe el ACK para confirmar a la interfaz gráfica la recepción del comando.

Por otro lado, para evitar el consumo de los periféricos no utilizados en este proyecto. Se retira la señal de reloj de los mismos. Para deshabilitar la señal de reloj se hace uso de un periférico llamado CLOCK_GATIN [22].
En CLOCK_GATIN encontramos tres registros de 32 bits donde cada bit corresponde a una señal de reloj de un periférico. La función implementada para dicho periférico, simplemente pone a 0 los bits enlazados al Clock de los periféricos no utilizados en este sistema.

Funciones

void	**CLK_gatin** (void)
	CLK_gatin función que deshabilita el reloj de los periféricos que no están en uso.

Tabla 7.Función HAL CLK_GATIN.

Para verificar la funcionalidad de estas técnicas, se han realizado medidas del consumo en tres situaciones distintas. Sin aplicar ninguna técnica, aplicando únicamente el retiro de las señales de reloj de los periféricos no empleados y aplicando tanto el WFI() como la retirada de las señales de reloj.

Técnica aplicada	Corriente consumida(A)	Potencia(W)
Ninguna	0.1753	0.9
CLK_GATIN	0.1530	0.785
CLK_GATIN+WFI	0.1252	0.643

Tabla 8.Resultados mediciones de consumo.

Se aprecia claramente la reducción del consumo. Destacar que el consumo para el WFI+ CLK_GATIN, se ha medido en el estado de reposo, con el microprocesador dormido, a la espera de la interrupción de llegada de mensaje nuevo en el puerto serie. Momento en el que atenderá la petición correspondiente, y el consumo, ascenderá al mostrado utilizando únicamente el CLK_GATIN.

Capítulo 3: PC

El fin último del sistema es el almacenaje en el PC de los datos para su posterior procesado.
Para completar el sistema se ha desarrollado una aplicación sobre Matlab que comentaré a continuación y una Interfaz Gráfica que da al usuario acceso directo a la aplicación de manera sencilla.

Antes de abordar estas cuestiones, comenzaremos analizando el protocolo de comunicación que permite el intercambio de mensajes entre la aplicación del ordenador y la del microcontrolador.

3.1 Protocolo de comunicación

Basado en el intercambio de paquetes, envía comandos desde la aplicación del ordenador a la capa de control del microcontrolador, dándole órdenes a este o simplemente para confirmar sucesos con el fin de dar feedback al usuario sobre el desarrollo de las distintas acciones.
Por ello, encontramos dos tipos de comandos, de acción o de seguimiento. Los comandos de acción, fuerzan al microprocesador a realizar alguna de las funciones de la HAL como por ejemplo, configurar los distintos periféricos. En contra, los comandos de seguimiento, confirman el éxito de acciones a lo largo del proceso como es el inicio de la comunicación seria asíncrona.
En la siguiente tabla se detallan los comandos utilizados.

comandos	
Connect	Confirma el establecimiento de conexión con el canal (Seguimiento).
Configuration	Inicia la función de configuración de los periféricos (Acción)
Start	Inicia la transmisión (Acción)
Stop	Detiene la transmisión (Acción)
Close	Finaliza la conexión y ejecuta un Reset en la placa (Acción)

Tabla 9.Comandos de proceso de comunicación [19].

Los comandos se codifican con valores numéricos de 8 bits. El comando Connect, se codifica con un 8, Configuration con un 9, Start con un 10, Stop con un 11 y Close con un 12.

Los paquetes transmitidos se construyen colocando inicialmente los comandos y posteriormente, el tamaño de dicho paquete. El único paquete con un tamaño distinto de dos, es el encabezado por el comando de configuración, ya que contiene la información de los parámetros claves sobre los que debe basarse la configuración de los periféricos. El tipo de conversión, desde conversores externos o internos, el número de canales a utilizar y la frecuencia de muestreo, considerando las limitaciones mentadas en el capítulo 2.
Los valores a transmitir en cada paquete se codifican en 8 bits, dando un problema en los valores de la frecuencia de muestreo que pueden pasar con creces el máximo valor introducido en 8 bits. Estos valores se codifican en Little-Endian y se transmiten en las

últimas 4 posiciones del paquete.
Para mejorar la comprensión de la construcción de los paquetes, se detalla un ejemplo de este último comando ya que es el más complicado.

Paquete transmitido: [9 8 1 1 64 66 15 0]
Los dos primeros valores ya han sido explicados, corresponden con el comando y el tamaño del paquete transmitido. Es importantes enviar el tamaño, para que la CPU pueda decodificar y almacenar correctamente el paquete, conociendo donde termina este.
El tercer parámetro es el tipo de conversión. En este caso, de valor 1, indica que la conversión la realizan los conversores internos. La otra opción posible es 2, donde se adquirirían los datos de un conversor externo a través de un SPI (opción planteada pero no testeada ni implementada en su totalidad).
El cuarto, corresponde al número de canales. En este caso es 1, se pueden solicitar hasta un valor máximo de 16.
Por último, los 4 números siguientes corresponden con la frecuencia de muestreo. En el ejemplo representan la frecuencia máxima posible, 1MHz, como podemos observar son cuatro valores de 8 bits desde el menos significativo al más significativo.

No se pueden enviar los comandos en el orden que se quiera. Lo primero, siempre es iniciar la conexión al canal serie asíncrono. Posteriormente, pueden realizarse dos opciones, cerrar la conexión serie o solicitar la configuración de los periféricos. Una vez se confirma la configuración, podemos iniciar el proceso de transmisión o cerrar la comunicación. Por último cuando se confirma la recepción de los paquetes y por tanto, el fin del proceso, podemos realizar una nueva transmisión o cerrar la comunicación serie.
Todas estas peticiones esperan la recepción de un ACK. Este consiste en la repetición del paquete recibido por la CPU. La aplicación de Matlab compara ambos paquetes para confirmar que el ACK es correcto y proseguir con el proceso o dar al usuario la información correspondiente.

En la Figura 4, se muestra un ejemplo de lo explicado.

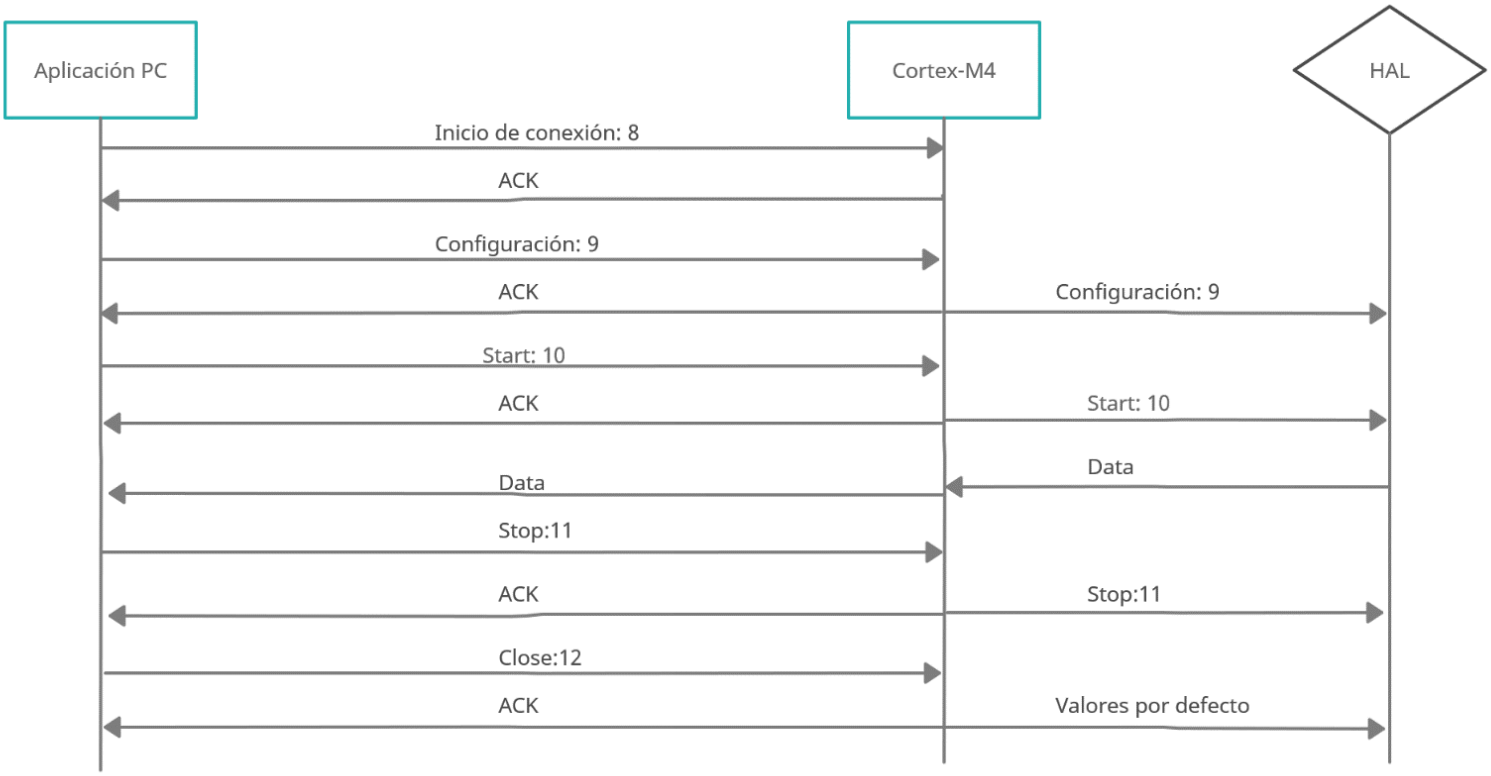

Figura 4. Ejemplo intercambio de mensajes [19].

3.2 Aplicación PC

Realiza todas las comunicaciones con el microcontrolador, la reconstrucción de los datos recibidos y la formación de los vectores proporcionados como salida del sistema. Para una mejor comprensión, se analizan las distintas funciones desde las fases del diagrama de flujo de la Figura 5.

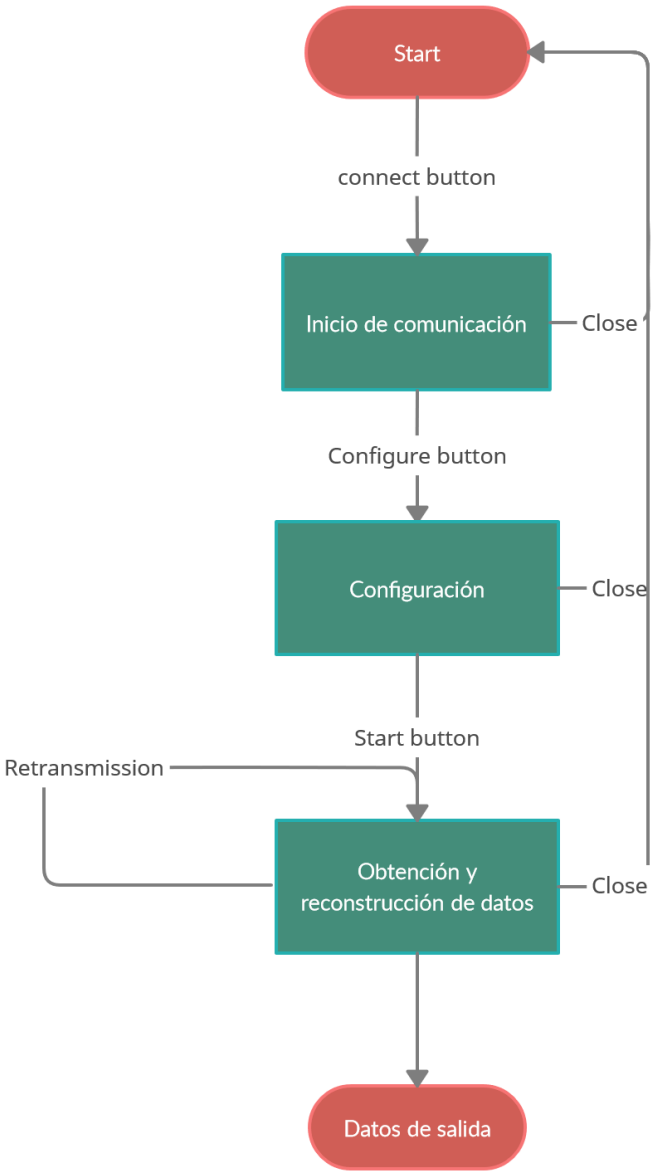

Figura 5. Diagrama de flujo de la aplicación [19].

Los estados son 4, Start, Inicio de Comunicación, Configuración y Reconstrucción de Datos.
El estado de Start, momento en el que se inicia la aplicación. Se obtienen los puertos COM disponibles para que el usuario pueda seleccionar el utilizado. Al pulsar el botón de la interfaz gráfica que indica la conexión, se cambia de estado al estado Inicio de Comunicación.

En el estado de inicio de comunicación, se establecen los parámetro del puerto serie utilizado, donde destacan el tamaño máximo de los buffer de recepción y transmisión, valor establecido a 96001, número máximo de transmisiones realizadas (48000

palabras de 16 bits son 96000 de 8 bits más los 8 bits de desbloqueo de la interrupción de transmisión del MFS0 dan los 96001 transmisiones) y el valor de la tasa de transmisión, se ha seleccionado un valor que garantice la estabilidad de las transmisiones, evitando que se produzcan pérdidas. Tras realizar distintas pruebas, se ha establecido en 100Kbaudios, valor que asegura la estabilidad de la comunicación.
Configurado el puerto serie, se envía el paquete correspondiente al inicio de la comunicación. La aplicación queda a la espera del correspondiente ACK y de las siguientes acciones del usuario.

El botón de configure, mueve el sistema al siguiente estado. Donde, se toman los valores deseados por el usuario en cuanto al tipo de conversión, frecuencia de muestreo y número de canales de conversión. Se forma el paquete, como se ha explicado en el punto anterior y tras enviarlo, queda a la espera de la recepción del ACK y las siguientes indicaciones del usuario.

Con el número de trasmisiones seleccionado se pulsa el botón de Start para pasar al siguiente estado, Obtención y reconstrucción de Datos. Es en este estado donde se generan los vectores de salida del sistema.
Se envía el paquete de Start y se espera a la recepción del ACK. Cuando se confirma el ACK correcto, la aplicación queda a la espera de la recepción del número de datos correspondiente, este número, depende del número de canales utilizados. Si únicamente se utiliza un canal, se recibirán 32001 valores para la reconstrucción. De haber 2, el número recibido es de 64001. Si se utilizan más de 2 canales se esperará recibir el número máximo de valores para su reconstrucción 96001.
Una vez recibidos, se envía el comando de Stop indicando la detención del proceso de conversión y transmisión. Se comienza en ese momento con la reconstrucción de datos.

Los datos convertidos en los ADCs son de 12 bits, se guardan en palabras de 16 bits, donde los 4 bits menos significativos no son accesibles para el diseñador [11].
El registro de transmisión del puerto serie, solo puede transmitir palabras de 8 bits, por lo tanto hay que partir la palabra original de 16 bits en dos de 8 bits y reconstruir, una vez se han transmitido.
Las transmisiones son en Little-Endian (se transmiten primero los bits menos significativos). Se almacenan todos los datos de 8 bits en un vector. Sin olvidar el primer uno transmitido para desbloquear la interrupción del puerto serie, se itera en un bucle desde la segunda posición del vector, evitando el primer uno recibido, realizando la siguiente operación.
Al valor actual, bits menos significativos, se le suma el valor siguientes, bits más significativos de los 16 bits obtenidos en los ADCs, multiplicado por 256. La multiplicación desplaza los 8 bits más significativos para colocarlos en su lugar correspondiente. Previamente a la suma, se han convertido los datos a 16 bits, los he llamado 8 bits menos significativos o más con el fin de que quede más clara la explicación.
Una vez tienes el dato completo de 16 bits, se trunca para obtener los 12 bits de datos.
Por último, el índice de iteración del bloque avanza de dos en dos para convertir el nuevo valor actual en el valor de los bits menos significativos del siguiente dato.

Una vez se tiene el vector definitivo, debe separarse en función del número de canales seleccionado.
El vector definitivo está formado, como máximo, por 48000 valores. 16000 datos de cada ADC. Como se ha dicho, los ADC van intercalando los datos de los distintos

canales. En función del número de canales que usa cada ADC se cogen los datos correspondientes a cada canal. Dando como resultado, un vector por canal utilizado.
Se escriben estos vectores en ficheros para realizar su posterior procesado, en el ejemplo de prueba que veremos en el capítulo siguiente solo se han mostrado los datos por pantalla.

Durante el desarrollo de cualquier punto de la aplicación, pulsar el botón Close, cierra la comunicación serie y hace volver a la aplicación al estado de Start.
Si se quiere volver a realizar una transmisión con la misma configuración. Debe volverse a pulsar el botón Start, realizando la acción de Retransmission. Esto cambiará los ficheros de los datos viejos, por otros ficheros con los nuevos datos. Si se quiere usar otra configuración, podría volverse a dar al botón configure con los nuevos parámetros deseados, pero es recomendable usar el botón Close ejecutando un Reset en el microcontrolador para asegurar que ningún periférico queda con una configuración indeseada.

3.3 Interfaz Gráfica de Usuario

Una interfaz gráfica de usuario, conocida también como GUI (del inglés Graphical User Interface), es un programa informático que interactúa con el usuario, haciendo uso de un conjunto de objetos gráficos para presentar la información y las acciones a realizar [23].

La interfaz gráfica ha sido desarrollada con la finalidad de dar al usuario un control sencillo sobre la aplicación del PC. Proporcionándole el control sobre las transiciones del diagrama de flujo de la APP.
Se ha utilizado appDesigner de matlab, que cuenta con un gran número de clases y procedimientos para incluir de manera rápida ventanas, botones, etiquetas y demás recursos gráficos que se usan habitualmente en las Interfaces Gráficas. Permitiendo un gran resultado pese a no ser una herramienta profesional [24].

Muestra los recursos gráficos con los valores de los parámetros que debe seleccionar el usuario para configurar el sistema.
También cuenta con recursos que dan información al usuario de cómo se están desarrollando las distintas acciones y de acciones que debe realizar. En la siguiente figura vemos como es la pantalla principal de la aplicación.

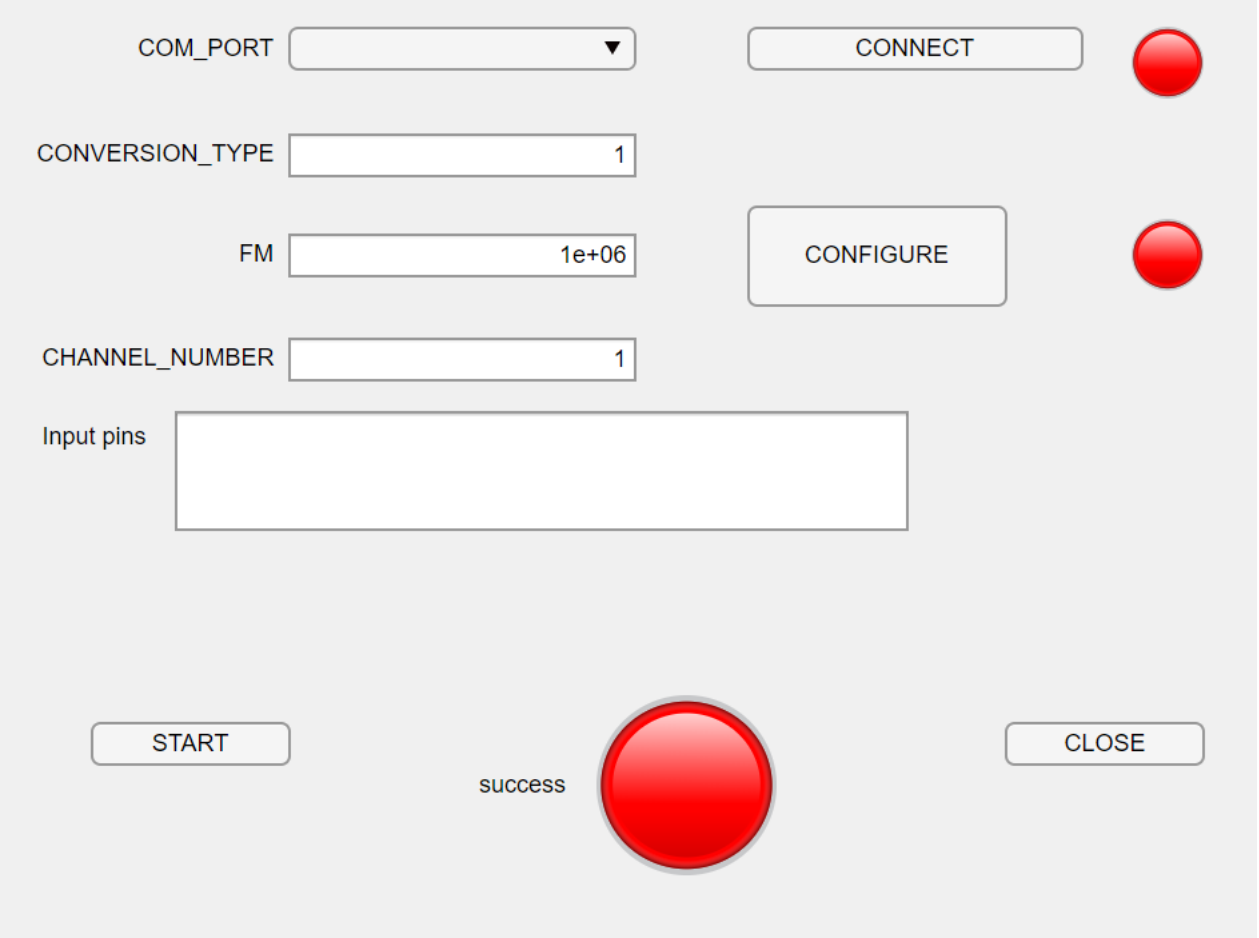

Figura 6. Interfaz Gráfica de usuario [19].

Vemos que se tiene control sobre cuatro botones, un desplegable y cuatro campos numéricos. Además, cuenta con tres leds que devuelven la información visual al usuario sobre lo realizado.

En el desplegable se cargarán, al iniciar la aplicación, los puertos COM disponibles en ese momento. Una vez seleccionado el puerto correspondiente, se debe pulsar el botón connect. Al hacerlo, se realiza la conexión con el puerto serie asíncrono del dispositivo. Para confirmar la conexión, se envía el comando pertinente y una vez recibido el ACK, se cambia el led de la pantalla de rojo a verde, informando al usuario de que se ha realizado la conexión con éxito.

Continúo comentando los distintos elementos sobre los que se tiene control siguiendo el proceso lógico de petición de datos.

Los campos numéricos, son editables por el usuario dentro del rango marcado para cada uno. Tres de ellos, corresponden con el tipo de obtención de datos, el número de canales y la frecuencia de muestreo de los ADCs. Estos son los valores que se envían al microcontrolador dentro del paquete de configuración. Una vez establecidos los valores deseados, se pulsa el botón de configuración, transmitiendo el paquete de configuración. Al recibir el ACK, si se enciende el led adyacente, la configuración se ha realizado correctamente. Además de esto, en función del número de canales seleccionados, se le indican al usuario, los pines a los que debe conectar las señales analógicas para realizar la conversión.

Tras colocar las señales en los pines indicados, se pulsa el botón de start dando comienzo al proceso de conversión y transmisión. Cuando ha concluido, se enciende el LED central, indicando que la transmisión ha sido un éxito.

Por último, el botón de cierre envía el comando para avisar al microprocesador de que

se va a proceder al cierre de la comunicación, momento en el que la CPU ejecuta un Reset para volver a los valores por defecto de los distintos periféricos. Al recibir el ACK, la aplicación cierra el canal serie y los recursos gráficos vuelven a su valor por defecto.

Capítulo 4: Evaluación funcional

Se ha realizado una batería de pruebas para probar el correcto funcionamiento del sistema.

Se han realizado distintas conversiones buscando comprobar los resultados obtenidos de convertir de todos los canales disponibles. Los vectores con los datos obtenidos simplemente muestran en una gráfica donde se demuestra que el resultado es el correcto.

Tanto ahora como a lo largo del proyecto, empero, no se ha tenido en cuenta la parte analógica de adaptación de la señal. No se han realizado los filtros pertinentes para eliminar ruido ni se han implementado etapas de ajuste de impedancias.
Al no considerar las impedancias de entrada del dispositivo físico y conectarlo directamente a los distintos generadores de señal, aparecen ciertas alteraciones en los resultados que serán comentadas más adelante.

En todas las pruebas, a excepción de la prueba de estabilidad donde se ha utilizado 1MHz, se ha utilizado una frecuencia de muestreo de 100 KHz, únicamente se han ido cambiando las señales de entrada. Para garantizar la estabilidad del sistema y no perder ningún dato, ya se ha dicho, que en la transmisión serie hay que establecer una tasa de 100 KBd. De no ser así, se producen pérdidas que provocan un desajuste a la hora de reconstruir los datos dando el resultado mostrado a continuación.

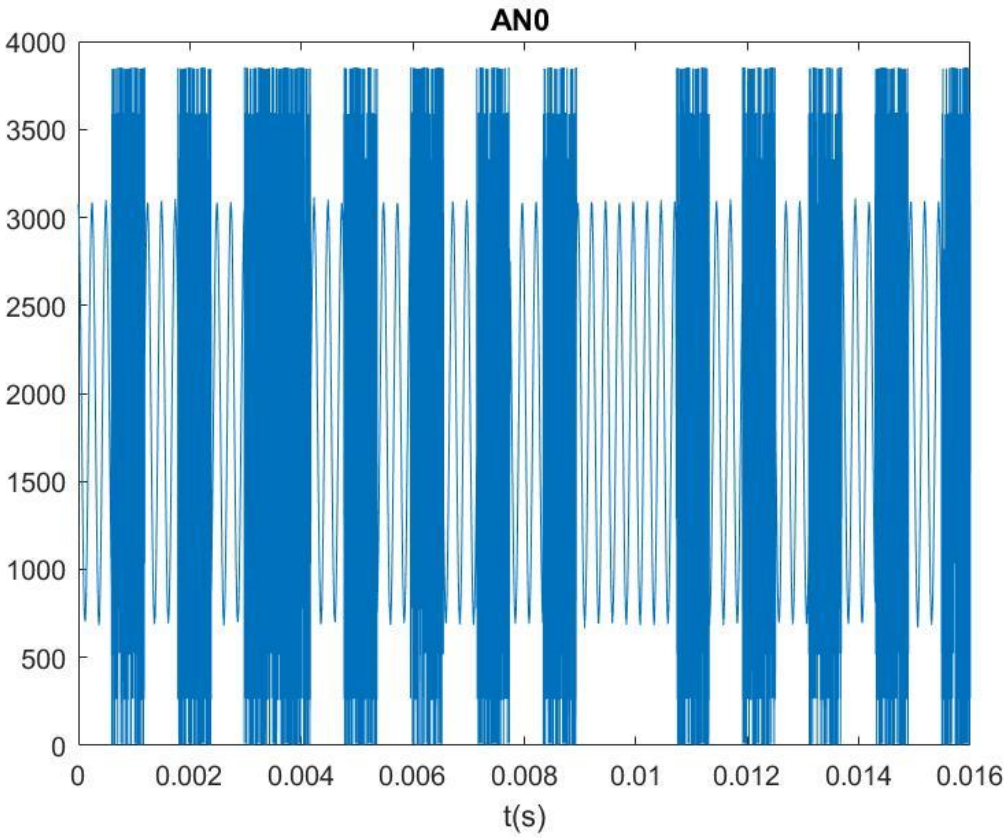

Figura 7. Resultado no estable.

Como se aprecia en la Figura7, en ciertos puntos la gráfica satura, esto se debe, empero, a la pérdida de alguna de las partes de las palabras de 16 bits. Por los motivos ya mentados, se transmiten palabras de 8 bits que deben reconstruirse en los datos convertidos de 12 bits. Al perder alguna de las palabras de 8 bits se producen los desajustes mostrados en la reconstrucción. Y no es hasta que se produce otra pérdida que no se corrige el problema.
Nótese que de perderse la palabra de 16 bits completa, no se apreciaría el fallo salvo en la cantidad de datos recibidos.
De ser necesaria una tasa de transmisión mayor, sería necesaria la implementación de un sistema de control de pérdidas.

También podemos comprobar el correcto funcionamiento de la frecuencia de muestreo. En este caso es de 1MHz, como el número de datos convertidos es fijo, 16000. Vemos que el tiempo de la señal es el esperado para dichos valores.
Resulta confuso que el tiempo fuera el correcto cuando se han producido pérdidas. Es correcto, ya que el estado del canal de DMA que transmite datos de la memoria externa al puerto serie, no se resetea hasta la recepción del comando de stop y este solo se genera al completar la transmisión del número de valores determinados. De tal forma, pese a perder valores y no completar la transmisión de tamaño determinada. El DMA vuelve a transmitir datos del inicio de la memoria hasta completar los datos esperados por la aplicación del ordenador y recibir el comando de stop.

Entendido que sucede al alterar la estabilidad del sistema y comprobada la veracidad de la frecuencia de muestreo. Comencemos con el análisis de las distintas pruebas.

Para comenzar, se ha introducido únicamente en un canal, un tono de 1KHz.

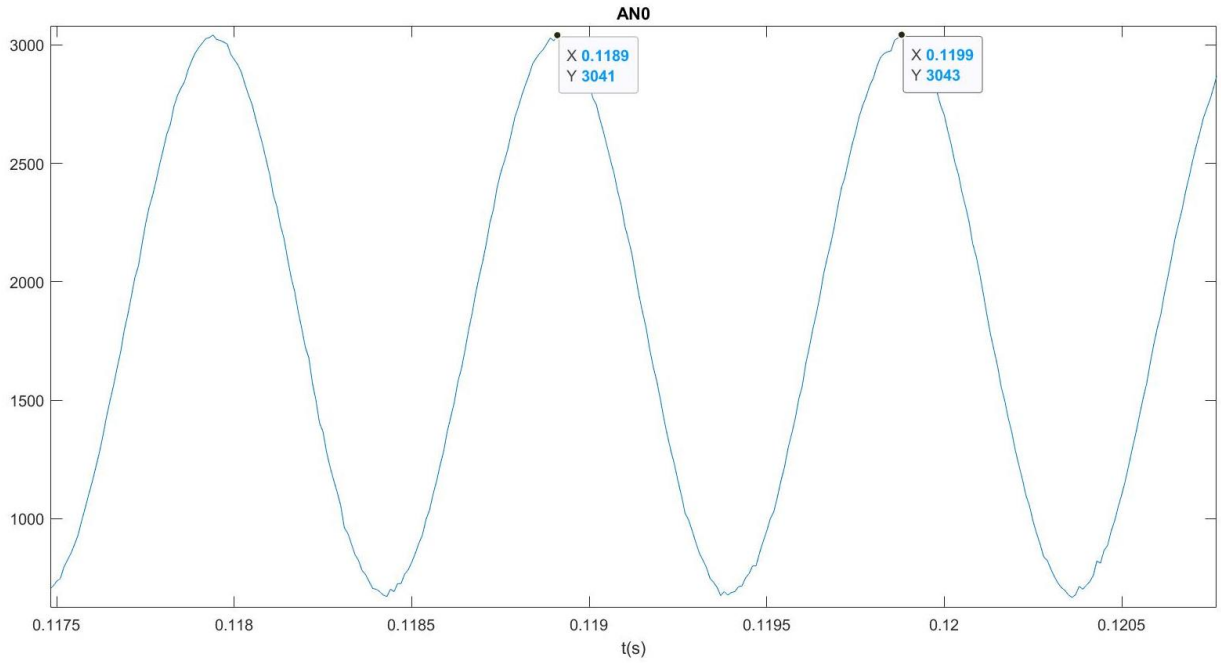

Figura 8. Tono de 1 KHz en AN0.

Para verificar que el resultado es correcto, se ha ampliado la señal mostrando que el periodo de la misma es de 0’001 segundos, correspondiente con una frecuencia de 1 KHz. Comprobamos por tanto que los conversores están realizando su trabajo correctamente, así como el resto de los eslabones del sistema.

Posteriormente, se han introducido 3 señales, un tono de 1KHZ y el mismo tono con dos divisores de tensión. Al usar resistencias idénticas en el divisor de cada canal, debíamos obtener una reducción de la tensión a la mitad. Pero al no tener en consideración las impedancias de entrada vemos que el resultado es más aproximado que concordante con la teoría. Además como en los divisores de cada uno de los canales se ha utilizado resistencias distintas, vemos que los resultados son ligeramente diferentes. Sea como fuere, para verificar el correcto funcionamiento de los ADCs es suficiente, además de ayudar a distinguir mejor las señales resultantes.

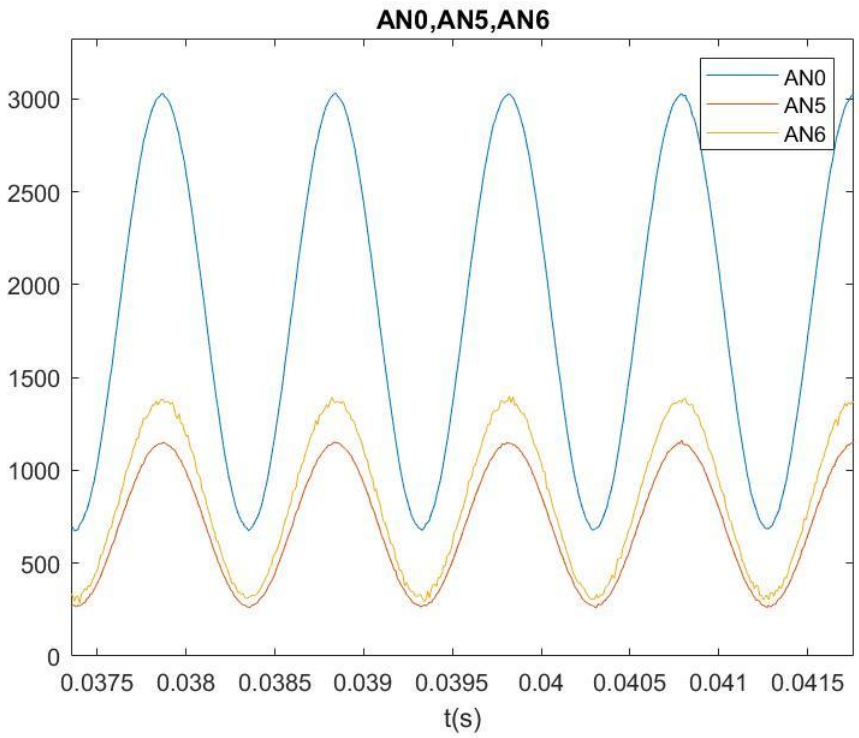

Figura 9. Prueba con 3 canales.

Vemos como el periodo si es idéntico pero la tensión resultante es aproximada.

La siguiente prueba, es la relativa al uso de 6 canales. Donde cada ADC convierte dos canales en vez de uno como en el caso anterior. Se produce por tanto una reducción a la mitad del número de datos convertidos para cada canal.

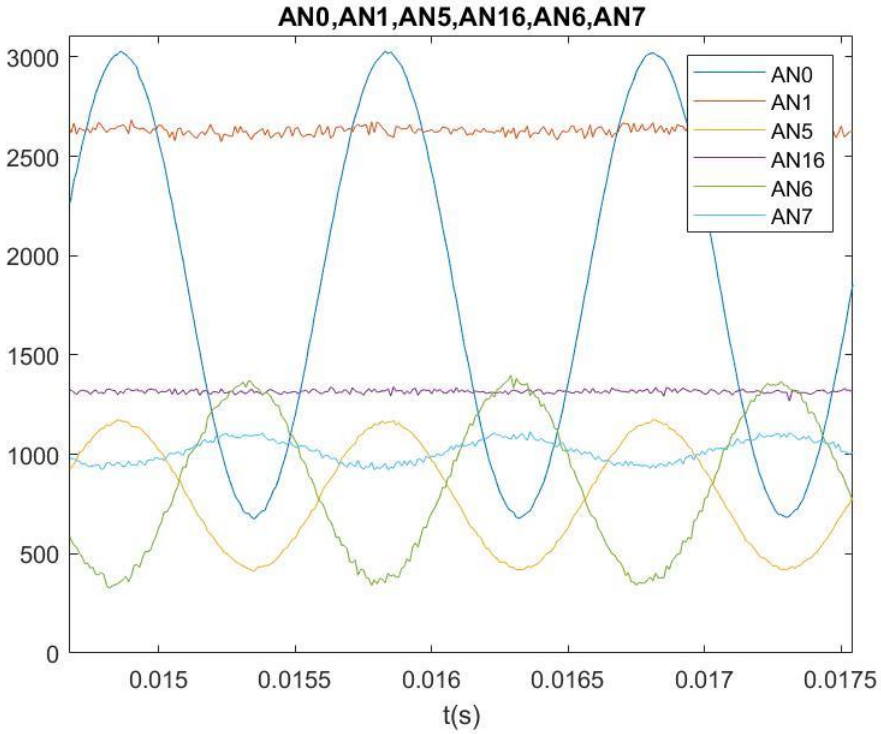

Figura 10. Prueba con 6 canales.

En los canales convertidos por el ADC0, el AN0 y el AN1, tenemos el tono de 1KHz y

una señal continua de 2V6. EL ADC1, canales AN6 y AN7 tiene un tono introducido desde otro generador y un divisor de la señal continua del canal AN1. Podemos ver como aparece una ligera modulación en dicha señal, aparecerá más adelante, y es debida a la impedancia de dicho canal.
Por último, el ADC2, convierte en el canal AN5 el tono de 1KHZ con un divisor de tensión y en el AN16, una señal continua obtenida de un divisor de tensión desde la señal continua de 2V6.

Para completar las pruebas, se han convertido con todos los canales posibles en esta placa, 14. Pese, como ya se ha dicho, a haber configurado el sistema para poder utilizar un máximo de 16 canales.
Con el fin de facilitar la comprensión de las gráficas mostradas, se han generado tres distintas, relativas a los canales que convierte cada uno de los ADCs, dejando claro el reparto de canales explicado en el punto anterior (referencia).

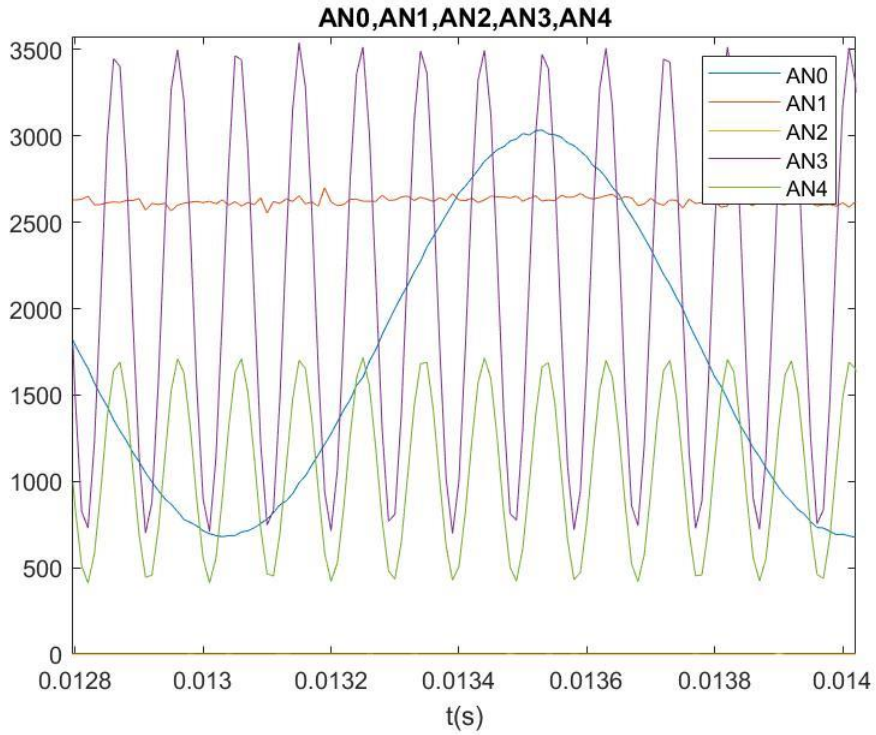

Figura 11. Prueba con 14 canales ADC0.

En este caso, tenemos el tono de 1KHZ en el AN0, otro tono de 10KHZ en los AN3 y AN4, solo que con un divisor de tensión a la entrada del AN4. La señal continúa de 2V6 en el canal AN1 y una señal de tierra en el canal AN2.

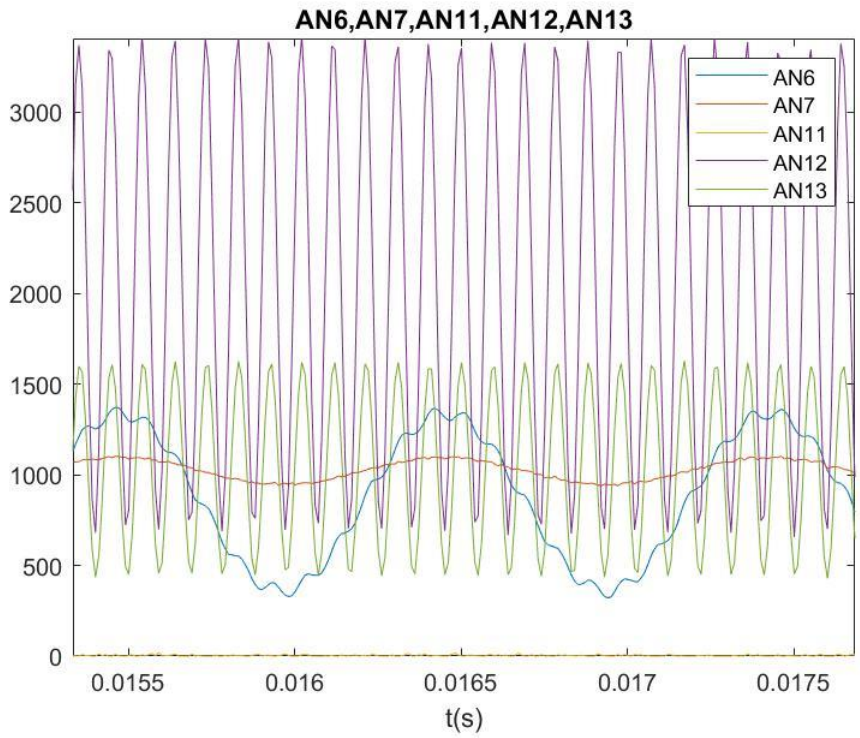

Figura 12. Prueba con 14 canales ADC1.

En el ADC1 tenemos las mismas señales, en los canales AN12 y AN13, que en los canales AN3 y AN4 respectivamente. La señal de tierra en el AN11 y en los canales AN6 y AN7, vemos que corresponden con las señales del AN0 y AN1 tras pasar por un divisor de tensión. Destacar la modulación que se aprecian en ambos casos, durante las pruebas, constantemente se ha achacado a ignorar las impedancias de entrada de los distintos canales, pero no se ha realizado un estudio detallado que permita asegurarlo.
Para descartar fallos, se comprobó que los valores almacenados en los espacios de memoria dedicados a esos datos, eran los mismos que los encontrados en los vectores divididos entre 4. Por lo tanto el error debe producirse en la conversión de los ADC, como en los otros canales este error no se produce, la ignorancia de las impedancias de entrada es una teoría probable, no siendo un fallo del sistema implementado.

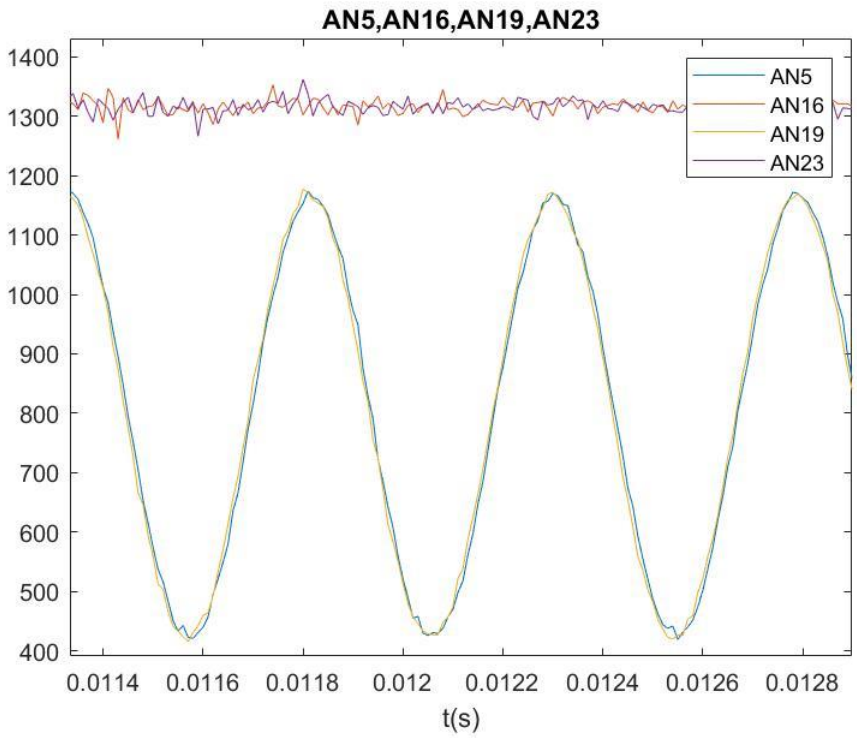

Figura 13. Prueba con 14 canales ADC2.

Por último, en el ADC2 se utilizan 4 canales. En los canales AN5 y AN16 se ha convertido una señal continua de 1V3. Y un tono de 200HZ en los canales AN19 y AN23.

Con estas pruebas queda comprobado el uso de todos los canales disponibles en la placa utilizada para este trabajo. Fue durante el desarrollo de las mismas que se corrigieron y se perfeccionaron todos los detalles del sistema, relativos tanto a las HAL que utilizan los periféricos de la placa, como a la aplicación del PC. Por lo tanto, empero destacar la importancia de las mismas a la hora de obtener un resultado final satisfactorio.

Capítulo 5: Conclusiones y líneas futuras de desarrollo

El proyecto ha concluido satisfactoriamente con la implementación de un sistema electrónico digital para la digitalización de señales analógicas.
Sistema programado, en distintas capas asegurando la independencia de las mismas y la portabilidad del código. Se han validado las técnicas utilizadas para la reducción del consumo, y se ha aportado una documentación clara de las distintas capas implementadas.

En lo relativo a la aplicación, se ha logrado la reconstrucción correcta de los datos recibidos por el microprocesador, mostrando una salida para cada canal utilizado. Destacar la GUI, que logra hacer del uso del sistema algo intuitivo para el usuario.

Las líneas de investigación y desarrollo que pueden dar lugar a una mayor amplitud del proyecto y que deberían tenerse en cuenta en posteriores actualizaciones están relacionadas con la mejora del protocolo de trasmisión de los datos, la implementación de diversos controles en el inicio de la conversión y el desarrollo de infraestructura analógica que permita una mejor conversión de las señales deseadas.

Para mejorar el protocolo de trasmisión, habría que incluir controles de pérdidas para evitar la pérdida de paquetes utilizando altas tasas de trasmisión. También plantear utilizar distintos protocolos permitidos por la placa, para enviar los datos al ordenador. Como puede ser, el uso de la salida Ethernet,o utilizar el puerto USB.

En cuanto al control de la conversión, incluir la posibilidad de realizar todo el proceso de conversión y trasmisión, disparando con una señal externa. Esto posibilitaría realizar las conversiones únicamente en el momento deseado por el usuario permitiendo que este sincronice la señal de inicio de conversión con la señal de excitación que provocará la reacción del tejido biológico.

El desarrollo de una estructura analógica, es referido a los distintos filtros aplicados a las señales antes de ser introducidas al sistema, con el fin de mejorar los resultados de este. Y añadir etapas para adaptar las impedancias de entrada de los distintos canales.

Además, por último, hay que configurar el puerto SPI para obtener los datos convertidos desde conversores externos y poder aprovechar las ventajas de las que disponen las placas específicas.

Bibliografía

[1] Intan Technologies, «RHD2000 Series Digital Electrophysiology Interface Chips,» 2013. [En línea]. Available: https://www.intantech.com/.

[2] Spansion, FM4 S6E2CC Series, 2014.

[3] ARM Developer, «Cortex -M,» [En línea]. Available: https://acortar.link/UMOTkx.

[4] J. Yiu, The Definitive Guide to ARM Cortex -M3 And Cortex -M4 Processors, Cambridge: Elsevier, 2014.

[5] Wikimedia, «Programación por capas,» [En línea]. Available: https://acortar.link/6mN10A.

[6] Tech Target, «Hardware Abstraction Layer,» [En línea]. Available: https://acortar.link/85ZEdi.

[7] R. Dr. E-Marmolejo, «Microcontrolador,» [En línea]. Available: https://hetpro-store.com/TUTORIALES/microcontrolador/.

[8] CMSIS, «Introducción a CMSIS,» [En línea]. Available: https://acortar.link/nNezzq.

[9] CMSIS, «CMSIS-Core,» [En línea]. Available: https://acortar.link/qnHDMl.

[10] Texas Instruments, «Analog to digital converters,» [En línea]. Available: https://acortar.link/irs9YS.

[11] ARM, FM4 32-bit Microcontroller FM4 Family Analog Macro Part, 2015.

[12] ARM, 32-bit Microcontroller FM4 Family Timer Part, 2015.

[13] Arquitectura de Computadoras, «Acceso Directo a Memoria,» [En línea]. Available: https://acortar.link/jB9ARp.

[14] ARM, FM4 32-bit Microcontroller FM4 family Peripheral Manual, 2015.

[15] Tech Target, «RAM,» [En línea]. Available: https://acortar.link/jxT6DF.

[16] Universidad de Santiago, «Adquisición de señales biológicas,» [En línea]. Available: https://acortar.link/KgOHWD.

[17] Tecnología del PC, «Puertos Serie,» [En línea]. Available: https://www.zator.com/Hardware/H2_5_1_1.htm.

[18] ARM, FM4 32-bit Microcontroller FM4 Family Communication Macro Part, 2015.

[19] Elaboración propia.

[20] Electrónica Básica Aplicada, «El Transistor Mosfet,» [En línea]. Available: https://acortar.link/Wx5lhs.

[21] ARM, Cortex -M4 Devices Generic User Guide, 2010.

[22] Semiconductor Ingenieering, «Clock Gatin,» [En línea]. Available: https://acortar.link/YugZ9m.

[23] Wikimedia, «Interfaz Gráfica de Usuario,» [En línea]. Available: https://acortar.link/8y1wIF.

[24] MathWorks, «APP Designer,» [En línea]. Available: https://acortar.link/qcE1Sl.

[25] Wikimedia, «Doxygen,» [En línea]. Available: https://en.wikipedia.org/wiki/Doxygen.

Anexo: Documentación

Para generar la documentación de la HAL del sistema se ha utilizado una herramienta muy extendida en el ámbito industrial, Doxygen. Doxygen es un generador de documentación de software en el que la documentación se genera automáticamente a partir del código. Doxygen incluye referencias cruzadas entre documentación y código, de modo que el lector puede saltar fácilmente entre el documento y el código [25].

1. HAL_ADC

◆ adc_clear()

```
void adc_clear ( void  )
```

adc_clear limpia las memorias de los ADC.

Definición en la línea **174** del archivo **ADC.c**.

◆ adc_config()

```
uint16_t adc_config ( uint16_t  n_canales,
                      uint32_t  frecuencia
                    )
```

adc_config Configura los ADC. Tener en cuenta que ambos parametros estan relacionados ya que los ADC reparten la frecuencia entre los canales activos. Por lo tanto, no se puede tener en un ADC dos canales activos a frecuencia maxima.El numero de canales va aumentando equitative entre los distintos ADCs, asi, podemos usar 3 canales a 1MHz, frecuencia maxima.

Parámetros

n_canales N de canales que quremos utilizar, cada ADC permite 5 canales y el ADC0 uno extra ,tener en cuenta que la frecuencia maxima del ADC es de 1MHz y que la frecuencia se divide entre los canales.

frecuencia Frecuencia a la que se quiere convertir.

Devuelve

El numero de ciclos que debe contar el Base Timer para establecer la frecuencia de muestreo deseada.

Definición en la línea **9** del archivo **ADC.c**.

◆ adc_start()

```
void adc_start ( void  )
```

adc_start inicia la conversion, debe colocarse en ultimo lugar.

Definición en la línea **166** del archivo **ADC.c**.

2.HAL_BaseTimer

◆ config_BT()

void config_BT (uint16_t cicles)

ini_BT configura el Base Timer.

Parámetros

cicles ciclos de reloj que debe contar para operar a la frecuencia deseada, los pasa la funcion config_adc

Definición en la línea 5 del archivo **BT.c**.

◆ start_BT()

void start_BT (void)

start_BT inicia el Base Timer

Definición en la línea 22 del archivo **BT.c**.

◆ stop_BT()

void stop_BT (void)

stop_BT detiene el Base Timer

Definición en la línea 18 del archivo **BT.c**.

3.HAL_DMA

◆ DMA_clear()

void DMA_clear (void)

Limpia el destino del DMA.

Definición en la línea 7 del archivo **DMA.c**.

◆ DMA_config()

void DMA_config (uint16_t TC,
uint32_t DR
)

DMA_config configura todos los canales de DMA a utilizar. Los canales de 0 a 2, este ultimo incluido, se usan para pasar datos desde los ADC a la memoria externa. El canal 3 se utiliza para transmitir los datos de la memoria al UART.

La configuracion es distinta en funcion de a quien se transmina. Para pasar los datos a la memoria externa,se utiliza el demand mode, este modo va cogiendo los datos de uno en uno y carga de golpe los 32Kbytes en la memoria. El canal 3 usa un modo de hardware block transfer, las peticiones se hacen tambien por hardware, igual que en el demand transfer, pero los datos se envian en paquetes para no sobreescribir la fifo del UART.

Parámetros

TC numero de transmisiones del canal 3 de DMA, en funcion del numero de ADCs convirtiendo datos.

DR direccion fuente del canal 1 del DMA, depende de los conversores utilizados.

Definición en la línea 16 del archivo **DMA.c**.

◆ DMA_stop()

void DMA_stop (void)

DMA_stop detiene los canales de DMA.

Definición en la línea **125** del archivo **DMA.c**.

◆ DMA_start()

void DMA_start (void)

DMA_start inicia los canales de DMA que quedan a la espera de las peticiones hardware para iniciar la transmision.

Definición en la línea **117** del archivo **DMA.c**.

4.HAL_UART

◆ uart_config()

void uart_config (void)

uart_config configura el transmisor y receptor del UART. Se utiliza la fifo con capacidad para 64 bytes aunque las transmisiones del DMA se haran en paquetes de 16 bytes(maximo permitido).

Definición en la línea **18** del archivo **UART.c**.

◆ canal_config()

void canal_config (void)

canal_config configura los pines que vamos a utilizar para crear la conexion puerto serie.

Definición en la línea **5** del archivo **UART.c**.

5.HAL_CLK_GATIN

◆ CLK_gatin()

void CLK_gatin (void)

CLK_gatin funcion que deshabilita el reloj de los perifericos que no estan en uso.

Definición en la línea **4** del archivo **CLK_GATIN.c**.

6.HAL_RAM Externa

◆ FM4_ram_init()

void FM4_ram_init (void)

FM4_ram_init inicializa la RAM externa para ser usada por el DMA.

Definición en la línea 9 del archivo FM4_ram.c.

7.Capa Media

◆ start()

void start (void)

start inicio de los perifericos del proyecto

Definición en la línea 39 del archivo macro_hal.c.

◆ stop()

void stop (void)

stop detencion de los perifericos del proyecto

Definición en la línea 47 del archivo macro_hal.c.

◆ configuracion_hal()

```
uint16_t configuracion_hal ( tipo     type,
                             uint16_t n_canales,
                             uint32_t frecuencia
                           )
```

configuracion_hal configura todos lo perifericos a utilizar en el proyecto

Parámetros

frecuencia frecuenciade funcionamiento de los ADC, se han implementado 4 en el tipo enumerado frez. Maxima frez 1MHz.

n_canales N de canales que quremos utilizar, cada ADC permite 16 canales,tener en cuenta que la frecuencia maxima del ADC es de 1MHz y que la frecuencia colocada en ini_adc se aplica en cada canal,con esto tratar de no pasar de 1MHZ

type tipo de conversion que se quiere realizar.

Devuelve

El numero de ciclos que debe contar el Base Timer, calculados por adc_config.

Definición en la línea 13 del archivo macro_hal.c.

8.Código

8.1. ADC.c

```
1. #include "mcu.h"
2. #include "ADC.h"
3. #include <stdint.h>
4.
5.
6.
7.
8.
9. uint16_t adc_config(uint16_t n_canales, uint32_t frecuencia){
10.
11.     uint16_t ciclos=0;
12.     ciclos= (1000000 * 199/ frecuencia);            //BT cicles
13.
14.//ADC0 configuration
15.
16.             FM4_ADC0->SCTSL = 0x02;         //Timer trigger Base timer ch.0
17.             bFM4_ADC0_SCCR_SHEN =1;                 //Timer start enable
18.             bFM4_ADC0_SCCR_SFCLR = 1;       //Clears FIFO
19.             bFM4_ADC0_ADCR_SCIE = 1;        //Interrupt request enable
20.             FM4_ADC0->ADST0 = 0x02; // Sampling time selection bits to 0.15us
21.             FM4_ADC0->ADCT = 0x03;    //  Comparation time selection bits to 0.35us
22.
23.
24.             if(n_canales<=3) {
25.
26.                     FM4_ADC0->SCIS0=0x01;           //Analog input channel 0
27.                     FM4_ADC0->SFNS = 0x0;   //sets up the generation of interrupt requests
28.
29.             }
30.             if(n_canales>3 & n_canales<=6){
31.
32.                     FM4_ADC0->SCIS0=0x03;           //Analog input channel 0 and 1
33.                     FM4_ADC0->SFNS = 0x0;           //sets up the generation of interrupt requests
34.
35.             }
36.             if(n_canales>6 & n_canales<=9) {
37.
38.                     FM4_ADC0->SCIS0=0x07;           //Analog input channel 0,1 and 2.
39.                     FM4_ADC0->SFNS = 0x0;           //sets up the generation of interrupt requests
40.             }
41.             if(n_canales>9 & n_canales<=12){
42.
43.                     FM4_ADC0->SCIS0=0x0F;   //Analog input channel 0,1,2 and 3.
44.                     FM4_ADC0->SFNS = 0x0;           //sets up the generation of interrupt requests
45.             }
46.             if(n_canales>12 & n_canales<=15){
47.
48.                     FM4_ADC0->SCIS0=0x1F;           //Analog input channel 0,1,2,3 and 4.
49.                     FM4_ADC0->SFNS = 0x0;           //sets up the generation of interrupt requests
50.             }
```

```
51.             if(n_canales==16) {
52.
53.                     FM4_ADC0->SCIS0=0x1F;           //Analog input channel 0,1,2,3 and4.
54.                     FM4_ADC0->SCIS1_f.AN8=1;//Analog input channel 8.
55.                     FM4_ADC0->SFNS = 0x0;           //sets up the generation of interrupt requests
56.             }
57.
58.
59. //ADC1 configuration
60.
61.
62.             FM4_ADC1->SCTSL = 0x02;         //Timer trigger Base timer ch.0
63.             bFM4_ADC1_SCCR_SHEN =1;                 //Timer start enable
64.             bFM4_ADC1_SCCR_SFCLR = 1;       //Clears FIFO
65.             bFM4_ADC1_ADCR_SCIE = 1;        //Interrupt request enable
66.             FM4_ADC1->ADST0 = 0x02; // Sampling time selection bits to 0.15us
67.             FM4_ADC1->ADCT = 0x03;    //  Comparation time selection bits to 0.35us
68.
69.             if(n_canales<=3 & n_canales>1) {
70.
71.                     FM4_ADC1->SCIS0_f.AN6=1;        //Analog input channel 6
72.                     FM4_ADC1->SFNS = 0x0;           //sets up the generation of interrupt requests
73.             }
74.             if(n_canales>4 & n_canales<=6){
75.
76.                     FM4_ADC1->SCIS0_f.AN6=1;        //Analog input channel 6
77.                     FM4_ADC1->SCIS0_f.AN7=1;        //Analog input channel 7
78.                     FM4_ADC1->SFNS = 0x0;           //sets up the generation of interrupt requests
79.
80.             }
81.             if(n_canales>5 & n_canales<=9) {
82.
83.                     FM4_ADC1->SCIS0_f.AN6=1;        //Analog input channel 6
84.                     FM4_ADC1->SCIS0_f.AN7=1;        //Analog input channel 7
85.                     FM4_ADC1->SCIS1_f.AN11=1;               //Analog input channel 11
86.                     FM4_ADC1->SFNS = 0x0;           //sets up the generation of interrupt requests
87.
88.             }
89.             if(n_canales>10 & n_canales<=12){
90.
91.                     FM4_ADC1->SCIS0_f.AN6=1;        //Analog input channel 6
92.                     FM4_ADC1->SCIS0_f.AN7=1;        //Analog input channel 7
93.                     FM4_ADC1->SCIS1_f.AN11=1;               //Analog input channel 11
94.                     FM4_ADC1->SCIS1_f.AN12=1;               //Analog input channel 12
95.                     FM4_ADC1->SFNS = 0x0;           //sets up the generation of interrupt requests
96.             }
97.             if(n_canales>13 ){
98.
99.                     FM4_ADC1->SCIS0_f.AN6=1;        //Analog input channel 6
100.                    FM4_ADC1->SCIS0_f.AN7=1;        //Analog input channel 7
101.                    FM4_ADC1->SCIS1_f.AN11=1;               //Analog input channel 11
102.                    FM4_ADC1->SCIS1_f.AN12=1;               //Analog input channel 12
103.                    FM4_ADC1->SCIS1_f.AN13=1;               //Analog input channel 13
104.                    FM4_ADC1->SFNS = 0x0;           //sets up the generation of interrupt requests
105.
```

```
106.    }
107.
108.
109.
110.
111.
112.
113.
114.    FM4_ADC2->SCTSL = 0x02;          //Timer trigger Base timer ch.0
115.    bFM4_ADC2_SCCR_SHEN =1;                  //Timer start enable
116.    bFM4_ADC2_SCCR_SFCLR = 1;        //Clears FIFO
117.    bFM4_ADC2_ADCR_SCIE = 1;         //Interrupt request enable
118.    FM4_ADC2->ADST0 = 0x02; // Sampling time selection bits to 0.15us
119.    FM4_ADC2->ADCT = 0x03;    //  Comparation time selection bits to 0.35us
120.
121.    if(n_canales==3) {
122.
123.        FM4_ADC2->SCIS0_f.AN5=1;         //Analog input channel 5
124.        FM4_ADC2->SFNS = 0x0;            //sets up the generation of interrupt requests
125.    }
126.    if(n_canales==6){
127.
128.        FM4_ADC2->SCIS0_f.AN5=1;         //Analog input channel 5
129.        FM4_ADC2->SCIS2_f.AN16=1;                //Analog input channel 16
130.        FM4_ADC2->SFNS = 0x0;            //sets up the generation of interrupt requests
131.
132.    }
133.    if(n_canales==9) {
134.
135.        FM4_ADC2->SCIS0_f.AN5=1;         //Analog input channel 5
136.        FM4_ADC2->SCIS2_f.AN16=1;                //Analog input channel 16
137.        FM4_ADC2->SCIS2_f.AN19=1;                //Analog input channel 19
138.      FM4_ADC2->SFNS = 0x0;              //sets up the generation of interrupt requests
139.    }
140.    if(n_canales==12){
141.
142.        FM4_ADC2->SCIS0_f.AN5=1;         //Analog input channel 5
143.        FM4_ADC2->SCIS2_f.AN16=1;                //Analog input channel 16
144.        FM4_ADC2->SCIS2_f.AN19=1;                //Analog input channel 19
145.        FM4_ADC2->SCIS2_f.AN23=1;                //Analog input channel 23
146.        FM4_ADC2->SFNS = 0x0;            //sets up the generation of interrupt requests
147.
148.    }
149.    if(n_canales>=15){
150.
151.        FM4_ADC2->SCIS0_f.AN5=1;         //Analog input channel 5
152.        FM4_ADC2->SCIS2_f.AN16=1;                //Analog input channel 16
153.        FM4_ADC2->SCIS2_f.AN19=1;                //Analog input channel 19
154.        FM4_ADC2->SCIS2_f.AN23=1;                //Analog input channel 23
155.        FM4_ADC2->SCIS2_f.AN17=1;                //Analog input channel 17
156.        FM4_ADC2->SFNS = 0x0;            //sets up the generation of interrupt requests
157.
158.    }
159.
160.
```

```
161. return ciclos;
162. }
163.
164.
165.
166. void adc_start(){
167.
168.      bFM4_ADC0_ADCEN_ENBL=1;              //Enables operation
169.      bFM4_ADC1_ADCEN_ENBL=1;              //Enables operation
170.      bFM4_ADC2_ADCEN_ENBL=1;              //Enables operation
171.
172.}
173.
174. void adc_clear(){
175.
176.
177.     if(bFM4_ADC0_ADCEN_ENBL==1){
178.
179.     bFM4_ADC0_SCCR_SFCLR=1;              //Clears FIFO
180.     bFM4_ADC0_ADCEN_ENBL=0;              //Disenables operation
181.     }
182.     if(bFM4_ADC1_ADCEN_ENBL==1){
183.
184.     bFM4_ADC1_SCCR_SFCLR=1;              //Clears FIFO
185.     bFM4_ADC1_ADCEN_ENBL=0;              //Disenables operation
186.     }
187.     if(bFM4_ADC2_ADCEN_ENBL==1){
188.
189.     bFM4_ADC2_SCCR_SFCLR=1;              //Clears FIFO
190.     bFM4_ADC2_ADCEN_ENBL=0;              //Disenables operation
191.     }
192. }
```

8.2. BT.c

```
1.#include "mcu.h"
2.#include "BT.h"
3.#include <stdint.h>
4.
5. void config_BT(uint16_t cicles){
6.
7.        FM4_BT0_PWM->TMCR_f.FMD0=1;             //Set FMD = "001" PWM function
8.        FM4_BT0_PWM->PCSR = cicles;  //Set cicles
9.        FM4_BT0_PWM->PDUT= cicles/2; //Set Duty
10.       bFM4_BT0_PPG_STC_UDIE = 1;              //Enable Interrupt Request
11. bFM4_BT0_PPG_STC_UDIR = 0;            //Clear underflow interrupt
12.
13.
14. }
15.
16.
17.
18. void stop_BT(){
19.       bFM4_BT0_RT_TMCR_CTEN = 0;              //Stop Base Timer
20 .}
21.
22. void start_BT(){
23.
24. bFM4_BT0_RT_TMCR_CTEN = 1;           //Timer Enable bit
25. bFM4_BT0_RT_TMCR_STRG  = 1;                  //Software trigger bit
26.
27.
28.}
29.
```

8.3. DMA.c

```
1.#include "mcu.h"
2.#include "DMA.h"
3.#include <stdint.h>
4.
5.
6.
7. void DMA_clear(){
8.
9.  FM4_DMAC->DMACB0 &= 0xFFF8FFFF;           //clean destination address
10.       FM4_DMAC->DMACB1 &= 0xFFF8FFFF;                 //clean destination address
11.       FM4_DMAC->DMACB2 &= 0xFFF8FFFF;                 //celan destination address
12.
13.
14.
15.}
16. void DMA_config(uint16_t TC, uint32_t DR){//mirar si puede mandar a memoria externa
17.
18.
19. FM4_DMAC->DMACR_f.DE=1;              //Enables the operations of all of the channels.
20. FM4_DMAC->DMACR_f.PR=1;              //Applies the rotation method to the priority order.
21.
```

```
22.
23. //DMA channel 0 configuration
24.
25.     FM4_DMAC->DMACSA0 = DR;                                 //Source address register LSB of SCFD
26.     FM4_DMAC->DMACDA0 =0x60000000;//Destination address register ExternalRAM
27.     FM4_DMAC->DMACA0_f.IS=37u;     //IDREQ[5] -> Scan conversion interrupt signal from A/D converter unit0.
28.     FM4_DMAC->DMACA0_f.TC=0x3E7F;         //Number of transfers -> 16000
29.
30.
31. //DMACB0 configuration
32.
33.     FM4_DMAC->DMACB0_f.MS=2u;     // Demand transfer mode
34.     FM4_DMAC->DMACB0_f.TW=1;      // Half-word (16 bits)
35.     FM4_DMAC->DMACB0_f.FS=1;      // Fixes the transfer source address.
36.     FM4_DMAC->DMACB0_f.RC=1;      // Enables the reload function of BC/TC.
37.     FM4_DMAC->DMACB0_f.RS=1;      // Enables the reload function of the transfer source address.
38.     FM4_DMAC->DMACB0_f.RD=1;      // Enables the reload function of the transfer destination address.
39.     FM4_DMAC->DMACB0_f.CI=1;      // Enables an interrupt to be issued upon successful completion of transfer.
40.     FM4_DMAC->DMACB0_f.EM=0;      // Does not clear DMACA:EB upon completion of the transfer.
41.
42.     bFM4_INTREQ_DRQSEL_ADCSCAN0 =1; //seleccionar la interrupción del adc0 para el DMA
43.
44.
45.
46. //DMA channel 1 configuration
47.
48.     FM4_DMAC->DMACSA1 = ((uint32_t)&FM4_ADC1->SCFD) +2u;     //Source address register LSB of SCFD
49.     FM4_DMAC->DMACDA1 =0x60007D00         ;         //Destination address register SRAM1
50.
51.     //DMACA1 configuration
52.     FM4_DMAC->DMACA1_f.IS=38u;     //IDREQ[6] -> Scan conversion interrupt signal from A/D converter unit1.
53.     FM4_DMAC->DMACA1_f.TC=0x3E7F;         //Number of transfers -> 16000
54.
55.
56. //DMACB1 configuration
57.     FM4_DMAC->DMACB1_f.MS=2u;     // Demand transfer mode
58.     FM4_DMAC->DMACB1_f.TW=1;      // Half-word (16 bits)
59.     FM4_DMAC->DMACB1_f.FS=1;      // Fixes the transfer source address.
60.     FM4_DMAC->DMACB1_f.RC=1;      // Enables the reload function of BC/TC.
61.     FM4_DMAC->DMACB1_f.RS=1;      // Enables the reload function of the transfer source address.
62.     FM4_DMAC->DMACB1_f.RD=1;      // Enables the reload function of the transfer destination address.
63.     FM4_DMAC->DMACB1_f.CI=1;      // Enables an interrupt to be issued upon successful completion of transfer.
64.     FM4_DMAC->DMACB1_f.EM=0;      // Does not clear DMACA:EB upon completion of the transfer.
65.
66.     bFM4_INTREQ_DRQSEL_ADCSCAN1 =1; //seleccionar la interrupción del adc0 para el DMA
67.     bFM4_ADC1_ADCR_SCIF = 0; // clear ADC0 interrupt
68.
69. //DMA channel 2 configuration
70.
71.     FM4_DMAC->DMACSA2 = ((uint32_t)&FM4_ADC2->SCFD) +2u;     //Source address register LSB of SCFD
72.     FM4_DMAC->DMACDA2 =0x6000FA00 ;               //Destination address register SRAM2
73.
74.     //DMACA2 configuration
75.     FM4_DMAC->DMACA2_f.IS=39u;     //IDREQ[7] -> Scan conversion interrupt signal from A/D converter unit2.
76.     FM4_DMAC->DMACA2_f.TC=0x3E7F;         //Number of transfers -> 16000
```

```
77.
78.
79. //DMACB2 configuration
80.       FM4_DMAC->DMACB2_f.MS=2u;      // Demand transfer mode
81.       FM4_DMAC->DMACB2_f.TW=1;       // Half-word (16 bits)
82.       FM4_DMAC->DMACB2_f.FS=1;  // Fixes the transfer source address.
83.       FM4_DMAC->DMACB2_f.RC=1;       // Enables the reload function of BC/TC.
84.       FM4_DMAC->DMACB2_f.RS=1;       // Enables the reload function of the transfer source address.
85.       FM4_DMAC->DMACB2_f.RD=1;       // Enables the reload function of the transfer destination address.
86.       FM4_DMAC->DMACB2_f.CI=1;       // Enables an interrupt to be issued upon successful completion of transfer.
87.       FM4_DMAC->DMACB2_f.EM=0;       // Does not clear DMACA:EB upon completion of the transfer.
88.
89.       bFM4_INTREQ_DRQSEL_ADCSCAN2 =1; //seleccionar la interrupción del adc0 para el DMA
90.       bFM4_ADC2_ADCR_SCIF = 0; // clear ADC0 interrupt
91.
92. //DMA channel 3 configuration
93.
94.       FM4_DMAC->DMACSA3 =0x60000000;         //Source address register RAM
95.       FM4_DMAC->DMACDA3 =(uint32_t) & FM4_MFS0->TDR ;  //Destination address register TDR
96.
97.       //DMACA3 configuration
98.
99.       FM4_DMAC->DMACA3_f.IS=0x2D; //IDREQ[13]->Sending interrupt signal from MFS ch.0 45u
100.      FM4_DMAC->DMACA3_f.BC=15u; //Number of transfer blocks
101.
102.      FM4_DMAC->DMACA3_f.TC=TC;     //Number of transfers -> 1999 / 3999 / 5999
103.
104.
105.      // DMACB3 configuration
106.  FM4_DMAC->DMACB3_f.MS=0;          //block transfer mode
107.
108.      FM4_DMAC->DMACB3_f.TW=0; //Transfer width: BYTE
109.      FM4_DMAC->DMACB3_f.FD=1; //Fixes the transfer destination address.
110.      FM4_DMAC->DMACB3_f.RC=1; //Enables the reload function of BC/TC.
111.      FM4_DMAC->DMACB3_f.RS=1; //Enables the reload function of the source destination address.
112.      FM4_DMAC->DMACB3_f.EM=0; //Does not clear DMACA:EB upon completion of the transfer
113.
114.      bFM4_INTREQ_DRQSEL_MFS0TX=1; // conectar al transmisor MFS0
115.
116.}
117. void DMA_start(){
118.
119.      bFM4_DMAC_DMACA0_EB=1;         //The operation of the relevant channel is enabled
120.      bFM4_DMAC_DMACA1_EB=1;         //The operation of the relevant channel is enabled
121.      bFM4_DMAC_DMACA2_EB=1;         //The operation of the relevant channel is enabled
122.
123.}
124.
125. void DMA_stop(){
126.      bFM4_DMAC_DMACA0_EB=0;         //The operation of the relevant channel is disenabled
127.      bFM4_DMAC_DMACA1_EB=0;         //The operation of the relevant channel is disenabled
128.      bFM4_DMAC_DMACA2_EB=0;         //The operation of the relevant channel is disenabled
129.      bFM4_DMAC_DMACA3_EB=0;         //The operation of the relevant channel is disenabled
130.}
```

8.4. UART.c

```
1.#include "mcu.h"
2.#include "UART.h"
3.#include <stdint.h>
4.
5. void canal_config(){
6.
7.          // configurar canal como UART
8.
9.
10.         FM4_GPIO->ADE_f.AN31 = 0;// pin externo como input/output
11.         FM4_GPIO->PFR2_f.P1          =          1;          // usar pin como input/output de un periférico
12.         FM4_GPIO->PFR2_f.P2          =          1;          // usar pin como input/output de un periférico
13.         FM4_GPIO->EPFR07_f.SIN0S =          0;          // Usar SIN0_0 como input pin de MFS ch.0 SIN
14.         FM4_GPIO->EPFR07_f.SOT0B          =          1;          // Usar SOT0_0 como pin de salida
15.
16. }
17.
18. void uart_config(){
19.
20. // SMR
21. FM4_MFS0->SMR_f.MD = 0;     // Operation mode 0 (asynchronous normal mode)
22. FM4_MFS0->SMR_f.SBL = 0;    // SBL: Stop bit length select bit (1 bit)
23. FM4_MFS0->SMR_f.BDS = 0;    // LSB first (The least significant bit is first transferred.)
24. FM4_MFS0->SMR_f.SOE = 1;    // Enables a serial data output
25. //SSR
26. FM4_MFS0->SSR= 1u <<7;                    // Clears the received error flag (PE, FRE, ORE).
27.
28. //ESCR
29. FM4_MFS0->ESCR_f.FLWEN = 0; // Disables hardware flow control
30. FM4_MFS0->ESCR_f.ESBL = 0 ; // Extension stop bit length select bit
31. FM4_MFS0->ESCR_f.INV = 0 ;  // NRZ format
32. FM4_MFS0->ESCR_f.PEN = 0 ;  // Disables parity
33. FM4_MFS0->ESCR_f.P = 0 ;    // Even-number parity
34. FM4_MFS0->ESCR_f.L = 0 ;    // 8-bit length
35.
36.
37.// se va a usar el clock interno.
38. FM4_MFS0->BGR  =          999; // 100Kbps
39.
40.
41. //SCR
42. FM4_MFS0->SCR_f.UPCL = 1;       // Programmable clear (UART reset)
43. FM4_MFS0->SCR_f.TXE = 1;       // Transmit enable
44. FM4_MFS0->SCR_f.RXE = 1;       // Transmit enable
45. FM4_MFS0->SCR_f.RIE = 1;                                    //Enables the received interrupt.
46. FM4_MFS0->SCR_f.TIE = 1;                                    //Enables a transmit interrupt
47.
48.
49. //FIFO
50. FM4_MFS0->FCR1_f.FSEL = 0;     //  Transmit FIFO:FIFO1; Received FIFO:FIFO2
51. //FM4_MFS0->FCR1_f.FTIE=1;                //          Enables the transmit FIFO interrupt.
52. FM4_MFS0->FCR0_f.FCL1=1;                 // FIFO1 is reset.
```

```
53. FM4_MFS0->FCR0_f.FE1=1;                    // Enables the FIFO1 operation.
54.
55.
56. }
57.
```

8.5. CLK_GATIN.c

```
1.#include "CLK_GATIN.h"
2.#include "mcu.h"
3.
4. void CLK_gatin(){
5.
6. //CKEN0->HCLK
7.      FM4_CLK_GATING->CKEN0_f.MFSCK1=0; //The bus clock input to the multi-function serial interface channel
8.      FM4_CLK_GATING->CKEN0_f.MFSCK2=0;  //The bus clock input to the multi-function serial interface channel
9.      FM4_CLK_GATING->CKEN0_f.MFSCK3=0;  //The bus clock input to the multi-function serial interface channel
10.     FM4_CLK_GATING->CKEN0_f.MFSCK4=0;  //The bus clock input to the multi-function serial interface channel
11.     FM4_CLK_GATING->CKEN0_f.MFSCK5=0;  //The bus clock input to the multi-function serial interface channel
12.     FM4_CLK_GATING->CKEN0_f.MFSCK6=0;  //The bus clock input to the multi-function serial interface channel
13.     FM4_CLK_GATING->CKEN0_f.MFSCK7=0;  //The bus clock input to the multi-function serial interface channel
14.     FM4_CLK_GATING->CKEN0_f.MFSCK8=0;  //The bus clock input to the multi-function serial interface channel
15.     FM4_CLK_GATING->CKEN0_f.MFSCK9=0;  //The bus clock input to the multi-function serial interface channel
16.     FM4_CLK_GATING->CKEN0_f.MFSCK10=0;//The bus clock input to the multi-function serial interface channel
17.     FM4_CLK_GATING->CKEN0_f.MFSCK11=0;//The bus clock input to the multi-function serial interface channel
18.     FM4_CLK_GATING->CKEN0_f.MFSCK12=0;//The bus clock input to the multi-function serial interface channel
19.     FM4_CLK_GATING->CKEN0_f.MFSCK13=0;//The bus clock input to the multi-function serial interface channel
20.     FM4_CLK_GATING->CKEN0_f.MFSCK14=0;//The bus clock input to the multi-function serial interface channel
21.     FM4_CLK_GATING->CKEN0_f.MFSCK15=0;//The bus clock input to the multi-function serial interface channel
22.
23.
24.     //CKEN1->PCLK1
25.
26.     FM4_CLK_GATING->CKEN1_f.QDUCK0=0;//Gates the bus clock input to the corresponding quad counter
27.     FM4_CLK_GATING->CKEN1_f.QDUCK1=0;//Gates the bus clock input to the corresponding quad counter
28.     FM4_CLK_GATING->CKEN1_f.QDUCK2=0;//Gates the bus clock input to the corresponding quad counter
29.     FM4_CLK_GATING->CKEN1_f.QDUCK3=0;//Gates the bus clock input to the corresponding quad counter
30.
31.     FM4_CLK_GATING->CKEN1_f.MFTCK1=0;//The bus clock input to the multi-function timer unit and the PPG
channel
32.     FM4_CLK_GATING->CKEN1_f.MFTCK2=0;//The bus clock input to the multi-function timer unit and the PPG
channel
33.     FM4_CLK_GATING->CKEN1_f.MFTCK3=0;//The bus clock input to the multi-function timer unit and the PPG
channel
34.
35.     FM4_CLK_GATING->CKEN1_f.BTMCK1=0;//The bus clock input to the base timer channel
36.     FM4_CLK_GATING->CKEN1_f.BTMCK2=0;//The bus clock input to the base timer channel
37.     FM4_CLK_GATING->CKEN1_f.BTMCK3=0;//The bus clock input to the base timer channel
38.
39.     //CKEN2->PCLK2
40.
41.     FM4_CLK_GATING->CKEN2_f.CECCK0=0;//Gates the bus clock input to the HDMI-CEC/Remote Control
Reception channel
42.     FM4_CLK_GATING->CKEN2_f.CECCK1=0;//Gates the bus clock input to the HDMI-CEC/Remote Control
Reception channel
43.
44.     FM4_CLK_GATING->CKEN2_f.I2SCK0=0;//Gates the bus clock input to the I2S Interface channel
45.     FM4_CLK_GATING->CKEN2_f.I2SCK1=0;//Gates the bus clock input to the I2S Interface channel
46.
```

```
47.     FM4_CLK_GATING->CKEN2_f.CANCK0=0;//Gates the bus clock input to the CAN controller channel
48.     FM4_CLK_GATING->CKEN2_f.CANCK1=0;//Gates the bus clock input to the CAN controller channel
49.     FM4_CLK_GATING->CKEN2_f.CANCK2=0;//Gates the bus clock input to the CAN controller channel
50.
51.     FM4_CLK_GATING->CKEN2_f.USBCK0=0;//Gates the bus clock input to the USB channel
52.     FM4_CLK_GATING->CKEN2_f.USBCK1=0;//Gates the bus clock input to the USB channel
53.
54.     FM4_CLK_GATING->CKEN2_f.SDCCK=0;//Gates the bus clock input to SD card interface.
55.
56. }
57.
58.
```

8.6. FM4_ram.c

```
1.#include "s6e2cc.h"
2.#include "FM4_ram.h"
3.
4. static void DonaldDelay( volatile uint32_t nCount)
5. {
6. for (; nCount > 0; nCount--);
7. }
8.
9. void FM4_ram_init(void)
10. {
11.  // Configura GPIO PINS como puertos EBIF
12.
13.  // Bus de direcciones MAD
14.  //  MAD(9:1) => P7(10:1)
15.  FM4_GPIO->PFR7 |= (0x01FF<<2);     // b0000_0111_1111_1100
16.  //  MAD(11:10) => P1(15:14)
17.  FM4_GPIO->PFR1 |= (3<<14);         // b1100_0000_0000_0000
18.  //    Comparten con entradas analógicas
19.  bFM4_GPIO_ADE_AN14 = 0u;
20.  bFM4_GPIO_ADE_AN15 = 0u;
21.  //  MAD(18:12) => P2(4:10)
22.  FM4_GPIO->PFR2 |= (0x7F<<4);      // b0000_0111_1111_0000
23.  //    Comparten con entradas analógicas
24.  bFM4_GPIO_ADE_AN24 = 0u;
25.  bFM4_GPIO_ADE_AN25 = 0u;
26.  bFM4_GPIO_ADE_AN26 = 0u;
27.  bFM4_GPIO_ADE_AN27 = 0u;
28.  bFM4_GPIO_ADE_AN28 = 0u;
29.  bFM4_GPIO_ADE_AN29 = 0u;
30.
31.  // Bus de datos MADATA
32.  FM4_GPIO->PFRA = 0xFFFF;
33.  //    Comparten con entradas analógicas
34.  bFM4_GPIO_ADE_AN02 = 0u;
35.  bFM4_GPIO_ADE_AN03 = 0u;
36.
37.  // Señales de control
38.  //MCSX0_0
39.  bFM4_GPIO_PFR7_PE = 1u;
40.  //MOEX0
41.  bFM4_GPIO_PFR6_P3 = 1u;
42.  //MWEX0
43.  bFM4_GPIO_PFR6_P2 = 1u;
44.  //MDQM0
45.  bFM4_GPIO_PFR0_P8 = 1u;
46.  //MDQM1
47.  bFM4_GPIO_PFR0_P9 = 1u;
48.  //MCLKOUT
49.  bFM4_GPIO_PFR0_PA = 1u;
50.
51.  // Extended Pin Function Setting Register 10/11 (EPFR10/11) assigns functions to external bus peripheral pins.
52.  FM4_GPIO->EPFR10 = 0xFFFFFFFF;
```

```
53.  FM4_GPIO->EPFR11 = 0x03FFFFFF;
54.
55.  // Configura Acceso RAM externa CY62147EV30LL-45B2XI
56.  // Reset External Bus for clean start.
57.      FM4_CLK_GATING->CKEN0_f.EXBCK = 0u;  // Clk_PeripheralClockDisable(ClkGateExtif);
58.      FM4_CLK_GATING->MRST0_f.EXBRST = 1u; // Clk_PeripheralSetReset(ClkResetExtif);
59.      FM4_CLK_GATING->MRST0_f.EXBRST = 0u; // Clk_PeripheralClearReset(ClkResetExtif);
60.      FM4_CLK_GATING->CKEN0_f.EXBCK = 1u;  // Clk_PeripheralClockEnable(ClkGateExtif);
61.  //
62.    DonaldDelay(10000);
63.  //
64.  FM4_EXBUS->MODE0=0; // clean
65.  FM4_EXBUS->MODE0=0xC85; //  (1<<10)|(1<<7)|(1<<2)|(1<<0);
66.     // bit 0: WDTH = 1u;     ///< 8, 16 bit data bus width. See description of #en_extif_width_t
67.     // bit 2: RBMON = 1u;    ///< TRUE: Read Byte Mask enable
68.     //        WEOFF = 0u;    ///< TRUE:Write enable disabled
69.     //        NAND = 0u;     ///< NAND Flash bus enable, FLASE: NOR Flash/SRAM bus enable
70.     //        PAGE = 0u;     ///< TRUE: NOR Flash memory page access mode enabled
71.    //        RDY = 0u;      ///< TRUE: RDY mode enabled
72.    //        SHRTDOUT = 1u; ///< TRUE: Stop to write data output at first idle cycle, FALSE: Extends to write
73.    //        MPXMODE = 0u;  ///< TRUE: Multiplex mode
74.   // bit 9: ALEINV = 0u;   ///< TRUE: Invert ALE signal (negative polarity)
75.   // bit11: MPXDOFF = 1u;  ///< TRUE: Do not output address to data lines (Hi-Z during ALC cycle period)
76.  //        MPXCSOF = 0u;  ///< TRUE: Do not assert MCSX in ALC cycle period
77.  // bit13: MOEXEUP = 0u;  ///< TRUE: MOEX width is set with FRADC, FALSE: MOEX width is set with RACC-RADC
78.  //
79.  DonaldDelay(10000);
80.  //
81.  FM4_EXBUS->TIM0=0; //clean
82.  FM4_EXBUS->TIM0_f.RACC = 4 - 1u;  ///< Read Access Cycle timing
83.  FM4_EXBUS->TIM0_f.RADC = 0;       ///< Read Address Set-up Cycle timing
84.  FM4_EXBUS->TIM0_f.FRADC = 1 - 1u; ///< First Read Address Cycle timing
85.  FM4_EXBUS->TIM0_f.RIDLC = 1 - 1u; ///< Read Idle Cycle timing
86.  FM4_EXBUS->TIM0_f.WACC = 5 - 1u;  ///< Write Access Cycle timing
87.  FM4_EXBUS->TIM0_f.WADC = 1 - 1u;  ///<Write Address Set-up Cycle timing
88.  FM4_EXBUS->TIM0_f.WWEC = 4 - 1u;  ///< Write Enable Cycle timing
89.  FM4_EXBUS->TIM0_f.WIDLC = 1 - 1u; ///< Write Idle Cycle timing
90.  //
91.  DonaldDelay(10000);
92.  //
93.  #define PSRAM_BASE_ADDRESS (0x60000000ul)
94.  //
95.  FM4_EXBUS->AREA0=0;
96.  FM4_EXBUS->AREA0_f.ADDR = (uint8_t)(PSRAM_BASE_ADDRESS >> 20ul); ///< Address bits [27:20]
97.  FM4_EXBUS->AREA0_f.MASK = 1;                                     //< See description of #en_extif_mask_t (2Mb)
98.  //
99.  FM4_EXBUS->ATIM0=0;
100.  FM4_EXBUS->ATIM0_f.ALC = 1 - 1u; ///< Address Latch Cycles
101.  FM4_EXBUS->ATIM0_f.ALES = 0;
102.  FM4_EXBUS->ATIM0_f.ALEW = 1 - 1u;
103.  //
104.  DonaldDelay(10000);
105.  //
106.  FM4_EXBUS->SDMODE = 0;          ///< SDRAm Enable
107.  FM4_EXBUS->REFTIM = 0;          ///< Refresh time
```

```
108.  FM4_EXBUS->PWRDWN_f.PDC = 0;    ///< u16PowerDownCount
109.  FM4_EXBUS->SDTIM = 0;          // RAS/CAS timing
110.  FM4_EXBUS->MEMCERR = 0;        ///< No interrupt
111.  FM4_EXBUS->DCLKR_f.MDIV = 3u;   ///< Division ratio for MCLK (1 ... 16 div)
112.  FM4_EXBUS->DCLKR_f.MCLKON = 1u; ///< TRUE: Enables MCLKOUT pin
113.  FM4_EXBUS->AMODE_f.WAEN = 0;    ///< TRUE: Enables preceding read and continuous write request
114.  //
115.  DonaldDelay(10000);
116.  //
117.}
```

8.7. Macro_hal.c

```
#include "mcu.h"
#include "ADC.h"
#include "DMA.h"
#include "UART.h"
#include "BT.h"
#include "macro_hal.h"
#include <stdint.h>

uint16_t tc=0x0; // Number of transfers DMA channel 3
uint32_t dr=0x0; // Source DMA address DMA channel 0
uint16_t ciclos; // Number of cycles for frequency

uint16_t configuracion_hal(  tipo type, uint16_t n_canales,uint32_t frecuencia){

	if(type==inter){
	adc_clear();
	ciclos=adc_config(n_canales,frecuencia);
	dr=((uint32_t)&FM4_ADC0->SCFD) +2u;
	if(n_canales>2){
		tc=0x176F;
	}
	if(n_canales==2){
		tc=0xF9F;
	}
	if(n_canales==1){
		tc=0x7CF;
	}
	}
	if(type==ext){
		if(n_canales>2){
		tc=0x176F;
		}
	}

	DMA_config(tc,dr);
 return ciclos;
}

void start(){

	DMA_start();
	adc_start();


}

void stop(){

	adc_clear();
```

```
50.     DMA_stop();
51.     stop_BT();
52. }
```